NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

SCHEMA AND DATA CONFLICT RESOLUTION ACROSS DISTRIBUTED GRAPHICAL ASN.1 DATABASES

by

Gino Celia, Jr.

September, 1994

Thesis Advisor: Magdi Kamel

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September, 1994	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE Schema and Data Conflict Resolution Across Distributed Graphical ASN.1 Databases (U)	5. FUNDING NUMBERS
6. AUTHOR Celia, Gino Jr.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE A

13. ABSTRACT *(maximum 200 words)* Recently, most large corporations, including the Department of Defense and the Department of the Navy have seen a dramatic proliferation of incompatible databases and their associated database management systems. Sooner or later, these organizations discover the need to integrate the data in these incompatible databases. One solution to this problem is the use of markup languages like Abstract Syntax Notation One (ASN.1) as a standard format for representing these daatabases and output reports and thus facilitating their integration. A main requirement of this integration approach is the ability to correctly identify and resolve the semantic conflicts that arise in the marked-up databases and outputs of software tools before any integration can take place. This thesis addresses this issue by introducing a systematic approach for identifying and resolving semantic conflicts for these databases and developing a prototype tool which aids in this resolution. We hope that this tool will greatly aid in the efforts of integration and manipulation of ASN.1 databases.

14. SUBJECT TERMS Distributed Databases, Conflict Resolution, ASN.1, Integration	15. NUMBER OF PAGES 126
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited.

Schema and Data Conflict Resolution Across Distributed Graphical ASN.1 Databases

by

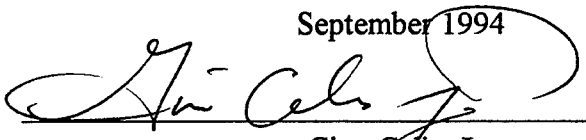
Gino Celia, Jr.
Lieutenant, United States Navy
B.S. (Computer Science), United States Naval Academy, 1988

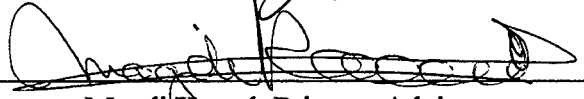
Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
September 1994

Author: 
Gino Celia, Jr.

Approved by: 
Magdi Kamel, Primary Advisor


James Emery, Associate Advisor


David R. Whipple, Chairman
Department of Systems Management

ABSTRACT

Recently, most large corporations, including the Department of Defense and the Department of the Navy, have seen a dramatic proliferation of incompatible databases and their associated database management systems. Sooner or later, these organizations discover the need to integrate the data in these incompatible databases. One solution to this problem is the use of markup languages like Abstract Syntax Notation One (ASN.1) as a standard format for representing these databases and output reports and thus facilitating their integration. A main requirement of this integration approach is the ability to correctly identify and resolve the semantic conflicts that arise in the marked-up databases and outputs of software tools before any integration can take place. This thesis addresses this issue by introducing a systematic approach for identifying and resolving semantic conflicts for these databases and developing a prototype tool that aids in this resolution . We hope that this tool will greatly aid in the efforts of integration and manipulation of ASN.1 databases.

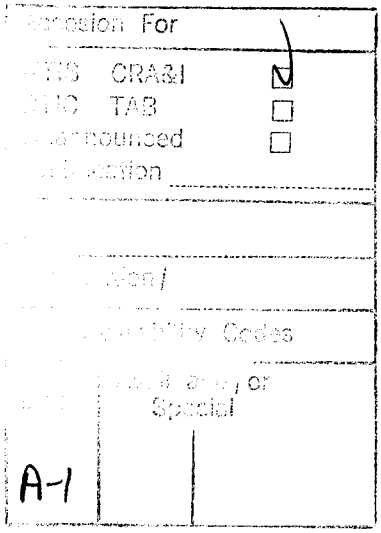


TABLE OF CONTENTS

ACKNOWLEDGMENTS

The author would like to express his thanks and gratitude to Professor Magdi Kamel, US Naval Postgraduate School, for his patience, guidance and understanding throughout the development and preparation of this thesis. His insight and expertise contributed enormously to the quality and merit of this work.

However, the successful completion of this thesis would not have been possible without the unwavering love and support from my wife, Kimberley, and the sacrifices she and my children made during a sometimes trying period of deadlines, travel, and very long hours.

I. INTRODUCTION

A. BACKGROUND

During the past three decades, most medium and large organizations have seen a dramatic proliferation of databases and their associated database management systems (DBMS's). While these databases proved to be useful in supporting their different activities, organizations soon discover the need to access and share data across independent systems. Unfortunately, such systems are often developed on vastly different, and incompatible, hardware and software environments. Until now, users wishing to integrate data from two or more systems found themselves tied to the original DBMS hardware and software constraints, with many of the integration efforts performed manually or not at all.

A new approach was proposed recently that advocates the use of markup languages as the basis for building integrated information resources that allow the users to access remote databases and software tools having heterogeneous formats in a uniform way (Kamel, 1994). The integration approach is based on modeling the input and output files for each database or software tool using a markup language having the power of a context-free grammar. The markup language chosen for this project is Abstract Syntax Notation One (ASN.1) for reasons discussed later in this thesis. ASN.1 acts as a Data

Definition Language (DDL) for the integrated databases and is used to define the formats of each input file and generated output report.

B. OBJECTIVES

A main requirement of this integration approach is the ability to identify and resolve the semantic conflicts that arise in the marked-up databases and outputs of software tools before any integration can take place. This thesis addresses the issue by introducing a systematic approach for identifying and resolving semantic conflicts for these databases and developing a prototype tool that aids in this resolution.

C. SCOPE, LIMITATIONS, AND ASSUMPTIONS

The scope of this project is limited to the factors affecting the conflict resolution problems of ASN.1 database integration. While conflict resolution is only a small part of the entire integration process, separate research addresses many of the other factors individually. While some of these other areas are discussed briefly here, they are presented only to the extent that they are necessary to provide a proper background discussion. The combination of this thesis with other ongoing work will help build a full-scale application encompassing the necessary characteristics for effective database integration.

This thesis assumes the reader is familiar with general database description terms and design considerations. A working knowledge of ASN.1 or other markup languages is not necessary, and a brief description of ASN.1 is included.

D. DEFINITIONS AND ABBREVIATIONS

The following are acronyms used in this thesis:

ASN.1 - Abstract Syntax Notation One (ISO 8824 and 8825)

DDL - Data Definition Language

DML - Data Manipulation Language

NIH - National Institutes of Health

NCBI - National Center for Biotechnical Information

FDB - Federated Database

CDB - Component Database

SQL - Structured Query Language

BNF - Backus-Naur Form

E. ORGANIZATION

This thesis is organized as follows. Chapter II presents an overview of ASN.1 as it relates to database integration and introduces the DDL and DML. Chapter III describes the sample databases designed to illustrate the various conflicts addressed in the thesis. Chapters IV and V present the conflict classification framework and resolution strategies, respectively. Chapter VI discusses the implementation of the tool, and acts as a User's Guide to the software. Chapter VII suggests possible areas for future research along with a discussion of lessons learned. The Appendix contains a full listing of all source code files related to the implementation.

II. OVERVIEW OF ASN.1 AND ITS USE FOR DATABASE INTEGRATION

This chapter presents an overview of the Abstract Syntax Notation One language and its use as a basis for integrating heterogeneous databases.

A. BACKGROUND

ASN.1 was originally developed as a data structure description language for use in data transfer across networks with different hardware/software configurations (NCBI, 1993). Later, ASN.1 gained popularity as a generic data transfer markup language for use in transferring data across heterogeneous networks and databases (Kamel, N., 1993). It was adopted by the National Institutes of Health (NIH) as the format for dissemination of its biological databases to users throughout the world. NIH's National Center for Biotechnology Information (NCBI) manages the periodic distribution of these databases. This recent interest in ASN.1 has led to the development of several software tools which aid in the manipulation, parsing, and transfer of ASN.1 documents and specifications. This interest has also prompted the development of applications which utilize ASN.1 documents.

As originally developed, ASN.1 is strictly a data description or data definition language. ASN.1 provides a method of tagging data fields with descriptive labels and organizing these fields into a distinct hierarchy. In this matter, complex data structures are

built by arranging simple data types (e.g., INTEGER, VisibleString, real, etc.) into complex tree-like structures. Its ability to describe complex data structures in a simple, text-based manner makes ASN.1 a prime candidate for use as a database description and transfer language among multiple heterogeneous databases. ASN.1, however, is strictly a data description language; it does not provide a means for data manipulation. In order to use ASN.1 as a database transfer language, some capability for basic database query and manipulation is needed. To answer this need, the ASN.1 Data Manipulation Language (ASN.1 DML) was recently proposed and is currently being implemented (Kamel, N., 1993). Together the DDL and DML provide a package for the effective representation and manipulation of heterogeneous data across multiple platforms.

The remainder of this chapter is arranged as follows. Section B describes the approach of using markup languages as database integration tools. Section C gives a detailed description of the DDL, while Section D describes the DML and how its used for the manipulation of ASN.1 databases.

B. ASN.1 AS A DATABASE INTEGRATION TOOL

In examining tools that aid in the integration of remote heterogeneous databases, two approaches have been generally accepted and developed, the tightly-coupled (federated database) and loosely-coupled (multidatabase) approaches (Sheth & Larson, 1990). In the tightly-coupled approach, a unified global schema is constructed from the underlying individual schemas of the component databases to be integrated. In the loosely-coupled approach, the component database schemas are not integrated into a global schema.

Rather, a method of performing queries on multiple databases is defined and developed to allow access to several or all the component databases simultaneously in a uniform way.

The major difference between the two approaches can best be demonstrated by how a user views the component databases under the two approaches. In the tightly-coupled approach, the user would be presented with a *single* super-schema or federated schema that represent the integration of all the underlying sub-schemas. The user need not be concerned with the component database schemas or the integration process; he treats the FDB as a single database and poses all query and data manipulation operations on that schema.

In the loosely-coupled approach, the user would be presented with each of the sub-schemas and a powerful data manipulation language or set of tools that allow him to perform queries on several or all the component databases. In this approach, the user needs to be knowledgeable about the structure of the sub-schemas in order to successfully perform queries and data manipulation.

While conceptually appealing and potentially useful, both approaches suffer from several drawbacks. In the federated database approach, the primary difficulty is in developing and maintaining the global schema. Additionally, this global schema becomes very sensitive to changes in the sub-schemas. Any change in the component database schemas requires re-integration of the local schemas into a new global schema. Also, a complex mapping between the federated and component schemas needs to be developed

and maintained in order to provide the necessary transparency to the user of the federated system.

While the multidatabase approach avoids this problem, its main difficulty is the development of a data manipulation language capable of operating on the component databases simultaneously. This language will also be very sensitive to sub-schema changes, and these changes may require modification or redesign of the data manipulation language. Additionally, in both approaches, only the *schemas* of the local databases are the components of the integration. A more powerful approach would integrate not only the schemas, but any output reports generated by the local DBMS's, as these are less subject to change over time.

Recently, a new approach was proposed to integrate not only the schemas of the component databases, but also their individual tools (application programs) and output reports (Kamel, 1994). Research has shown that while the individual schemas of the component databases are subject to fairly frequent change, the heavy public dependence on their related *tools and reports* puts pressure on administrators not to change these items frequently. The proposed approach is based on using a markup language to perform the integration of the component database schemas, tools, and/or output reports to provide either a tightly-coupled or loosely-coupled multidatabase system that is less sensitive to changes in the underlying sub-schemas than any system previously developed.

To accomplish integration using either approach, the issues of identifying and resolving semantic conflicts needs to be addressed. The primary goal of this thesis is to

address the schema and data conflict resolution strategies for integrating these ASN.1 documents.

C. ASN.1 DATA DEFINITION LANGUAGE (DDL)

Standard ASN.1 is a notation for defining abstract data types and their values. These data types can be broadly classified into simple types, structured types, and other types. *Simple* types are atomic types with no components, and include Boolean, integer, real, enumerated, and a variety of character string types. *Structured* types, also known as constructors, consist of four types for building complex data types from simple data types. *Other* types include the CHOICE and ANY data types.

ASN.1 is used to both describe the complex data structure and specify the data values. An ASN.1 document that describes the data structure is referred to as an ASN.1 *specification*, while a document that contains the data is known as an ASN.1 *printfile*. ASN.1 documents (both specifications and printfiles) follow a strict format of sequences of <**identifier, value**> pairs. Identifiers are tags that are user-defined and usually help describe the value object. In an ASN.1 specification, values are the type of the identifier, whereas in a printfile, values are the actual data values. In either case, the value may be a complex data type known as a type reference, which is described in a separate ASN.1 specification. A sample specification and part of its associated printfile are given in Figures 1 and 2. A detailed description of this database is given in Chapter III.

```
Holding ::= SEQUENCE {
       b-num INTEGER,                              -- local key
       title VisibleString,
       author-name VisibleString,                  -- last, first
       subj VisibleString OPTIONAL,
       type CHOICE {
              book Book-type,
              music Music-type,
              movie Movie-type },
       language VisibleString DEFAULT "English",
       lc-num SEQUENCE {
              c-letter VisibleString,              -- one+ CAP ltrs
              f-digit VisibleString,               -- one or more digits
              s-digit VisibleString OPTIONAL,      -- one or more digits
              cuttering VisibleString },           -- auth cutter number
       publisher-name VisibleString,
       publisher-addr VisibleString,               -- num, str, city, st
       checked-out BOOLEAN,                        -- TRUE if in library
       cost INTEGER }                              -- cost(whole dollars)
```

Figure 1. Sample ASN.1 Specification

```
Holding ::= {
  b-num 10 ,
  title "Joint Military Operations: A Short History" ,
  author-name "Beaumont, Roger A." ,
  subj "Military Science" ,
  type
    book {
      binding hardcover ,
      num-pgs 245 } ,
  language "English" ,
  lc-num {
    c-letter "U" ,
    f-digit "260" ,
    cuttering "B43" } ,
  publisher-name "Greenwood Press" ,
  publisher-addr "Westport, Connecticut" ,
  checked-out TRUE ,
  cost 60 }
```

Figure 2. Sample ASN.1 Printfile

In the above sample specification, a complex data structure is defined to describe the holdings of a library's database. In the printfile, a specific instance of a holding is shown with the values of appropriate data fields specified. Note that this specification is for a

single holding only--if a group of holdings were to be represented, a new specification for a holding-set must be included. This specification would allow for a printfile that contained a SET or SEQUENCE of many holdings.

Since ASN.1 was originally developed for data transfer across networks (which also includes binary transfer), several data types would be redundant or unnecessary when applied to a text-based database description. For this reason, and for the sake of simplicity, we have limited the constructs and data types used in this project to the subset shown in Table 1. The sample databases developed in the following chapter are encoded using these constructs and data types. However, it may become necessary later to expand the data types and constructs chosen to include a wider variety.

TABLE 1
ASN.1 BASE TYPES AND DEFINITIONS

Type	Description	Specification	Printfile Notation
BOOLEAN	Any TRUE or FALSE value. May have a DEFAULT	Truth ::= BOOLEAN	Truth ::= FALSE
INTEGER	Any integer value. May be given named values but range not limited to names. May have a DEFAULT.	Number ::= INTEGER or Number ::= INTEGER { red(1), blue(2) }	Number ::= 42 or Number ::= red
OCTET STRING	Any string of bytes. May not have DEFAULT.	Hstring ::= OCTET STRING	Hstring ::= '0A01FH
NULL	null is only allowed value	Nothing ::= NULL	Nothing ::= null
REAL	Floating point number in base 2 or 10. REAL value notation is 3 integers for { matissa, base, exponent } May have a DEFAULT.	Pi ::= REAL	Pi ::= { 314159, 10, -5 }

ENUMERATED	A named set of integer values. Only named values allowed. May have a DEFAULT.	Sex ::= ENUMERATED{ male (1), female (2) }	Sex ::= male
SEQUENCE	A series of other named types, in order. All elements must be present unless OPTIONAL or DEFAULT.	Yuppie ::= SEQUENCE { income INTEGER, name VisibleString }	Yuppie ::= { income 100000, name "John Doe" }
SEQUENCE OF	A repeating series of a single type in order.	Stooges ::= SEQUENCE OF VisibleString	Stooges ::= { "Larry", "Curly", "Moe" }
SET	A series of other named types, order does not matter. All elements must be present unless OPTIONAL or DEFAULT.	Yuppie ::= SET { income INTEGER, name VisibleString }	Yuppie ::= { income 100000, name "John Doe" }
SET OF	A repeating series of a single type. Order does not matter.	Stooges ::= SET OF VisibleString	Stooges ::= { "Larry", "Curly", "Moe" }
CHOICE	A way to select one from a set of alternate types. NOTE: In the printfile notation, you are indicating one choice, so {} are not allowed but the identifier for the selected CHOICE must be given before the value	Person ::= CHOICE { ssn INTEGER, name VisibleString, badge-id INTEGER }	Person ::= name "Joe"
VisibleString	A string of printable ASCII characters. NOTE: The double quote character (") may be included in a VisibleString by doubling it.	Text ::= VisibleString	Text ::= "Hi Mom!"
StringStore	Defines a VisibleString which is read into a ByteStore instead of a CharPtr. Used for very long strings.	Dna ::= StringStore	Dna ::= "AGGAGG"

As indicated by the ASN.1 specification and printfile of Figures 1 and 2, ASN.1 structures (specifications and printfiles) are hierarchical in nature. By adopting a simple graphical notation, ASN.1 structures can be represented in a tree or hierarchical format.

With this format, a user can easily manipulate the on-screen structures using normal tree manipulation functions (pruning, combining, etc.) as well as standard database manipulation operators. Figure 3 shows the graphical representation of selected ASN.1 constructs . The graphical tree representation of the sample specification and printfile given in Figures 1 and 2 is shown as Database 1 in Figure 4.

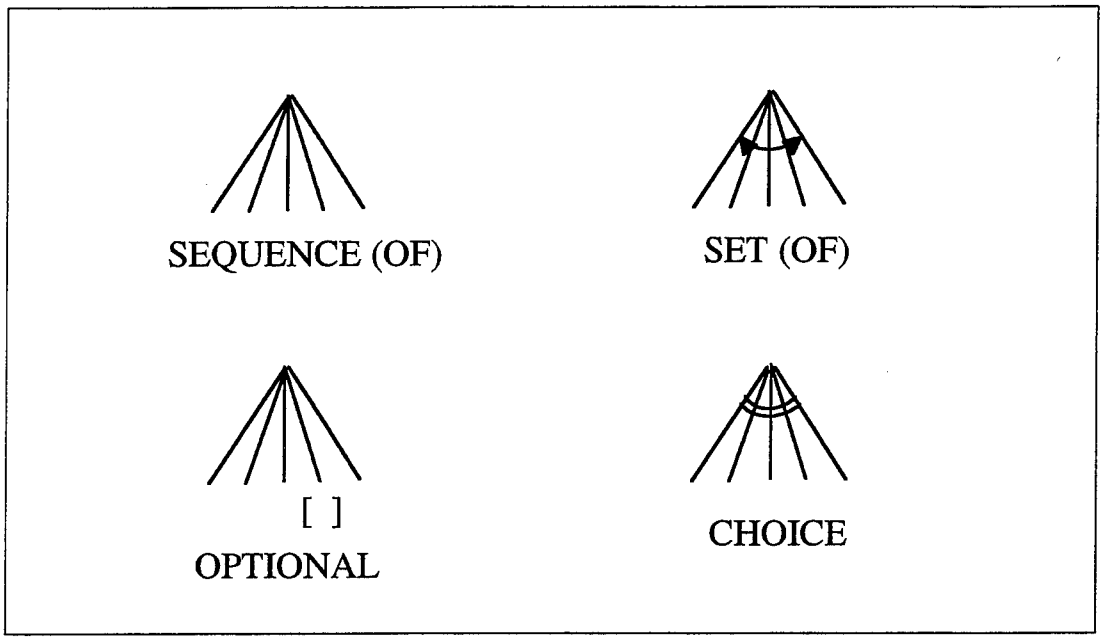


Figure 3. Graphical Representation of Supported ASN.1 Constructs

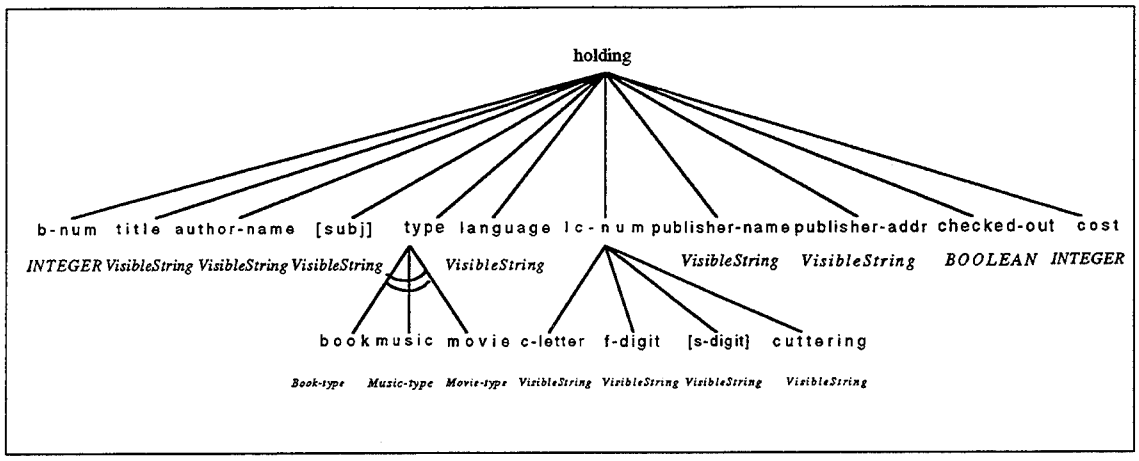


Figure 4. Sample ASN.1 Tree Structure

D. ASN.1 DATA MANIPULATION LANGUAGE (DML)

In anticipation of the data manipulation facilities required for applying ASN.1 to heterogeneous database data sharing, an SQL-like data manipulation language was developed (Kamel, N., 1993). This DML provides functions that allow normal database query and manipulation on ASN.1 printfiles. Some of these functions include subtree extraction, assignment, comparison, joining, importing/exporting, and rearranging subtrees. Since the DML is not part of the ASN.1 standard, it has not yet gained widespread use. Additionally, research is still ongoing to refine and redefine certain aspects of the DML. While the goal of this thesis is the resolution of schema and data conflicts and not data manipulation, conflict resolution should be seen as a necessary first step in developing database views (either loosely or tightly coupled) with all types of conflicts resolved and on which the user will perform DML processes. Some DML functionality is provided in the software developed in this thesis for demonstration

purposes and to complement the conflict resolution capabilities developed. The BNF syntax for the DML functions is shown in Table 2 (Kamel, 1994). It should be emphasized that DML functions are provided explicitly for the manipulation of the data in an ASN.1 database, much like SQL manipulates data in relational databases. These are not to be confused with the tools provided by this thesis which allow the user to manipulate the schema and data definitions in order to resolve integration conflicts. These tools will be discussed in more depth in Chapters IV and V.

TABLE 2
BNF SYNTAX FOR ASN.1 DATA MANIPULATION LANGUAGE

BNF syntax description of ASN.1 DML

<ASN.1_program> ::= **BEGIN** {*<statement>*;}...**END**

<statement> ::= {*<assignment_statement>* | *<GET_statement>* | *<IMPORT_statement>* | *<REIMPORT_statement>* | *<SHOW_statement>* | *<SET_statement>*} | *<ALTER_statement>*

<assignment_statement> ::= *<external_tree_name>* = *<external_tree_name>*

<GET_statement> ::= **GET** { **FIRST** | **ALL** [**UNIQUE**] } [*<subtree>*] {[, [*<subtree>*]]}...[**INTO** *<tree_name>*]
FROM {*<tree_name>* [INCLUDE PATH]}...
[**WHERE** *<condition>*]

<condition> ::= [({*<term>* | *<term>* { **AND** | **OR** } *<condition>* })]

<term> ::= { (*<comparison>*) | **NOT** *<comparison>*}

<comparison> ::= { **T** | **F** | *<factor>* *<op>* *<factor>*}

<factor> ::= {*<variable>* | *<constant>* }

<op> ::= { = | <> | > | < | >= | <= }

<IMPORT_statement> ::= **IMPORT** *<text_file>* **USING** *<abstract_specification>* **INTO** *<print_form>*

<REIMPORT_statement> ::= **REIMPORT** *<text_file>* **USING** *<abstract_specification>* [**INTO** *<print_form>*]

<EXPORT_statement> ::= **EXPORT** *<print-form>* **TO** *<text_file>*

<SHOW_statement> ::= **SHOW** { **ENV[IRONMENT]** *<text_file>* | *<abstract_specification>* | *<print-form>* }

<SET_statement> ::= **SET** *<system_variable>* [= *<value>*]

<ALTER_statement> ::= **ALTER** *<print-form>* **BY** *<abstract_specification>* [**INTO** *<print-form>*]

<text_file> ::= identifier

<abstract_specification> ::= identifier

<print-form> ::= identifier

<variable> ::= identifier

<system_variable> ::= {**PAUSE** | **LINE_WIDTH** | **PAGE_SIZE** | **SPOOL_FILE** | **DISPLAY_RESULTS** }

<value> ::= { integer | real | boolean | string }

<constant> ::= { integer | real | boolean | string }

III. SAMPLE DATABASES

A. SELECTION CRITERIA

In selecting sample databases to demonstrate semantic conflict identification and resolution for this thesis, several factors were taken into consideration. Initially, we intended to utilize some of the biological databases publicly distributed by the National Center for Biotechnical Information (NCBI). However, upon close examination it was determined that these databases would not yield enough conflicts to demonstrate our approach of conflict identification and resolution. Rather than modify the NCBI databases to meet our needs, we adapted two sample relational databases from Kim and Seo's paper on classifying conflicts in multidatabase systems (1991) and presented them in ASN.1 formats. These databases, with some modification, had the benefit of containing a majority of the conflicts we needed to illustrate our approach.

B. COMPONENT DATABASES

The component databases represent two independent library DBMS's, each in a different location, and implemented with different hardware and software. The following sections describe each database in the ASN.1 specification format, the ASN.1 printfile format, and graphically.

1. Database 1--Main Library

The first sample database represents the holdings of the Main Library and is shown as an ASN.1 specification in Figure 5.

```
--****************************************
--
--  CDB-1 data definitions
--  Gino Celia, 1994
--
--****************************************

Component-one-module DEFINITIONS ::=
BEGIN

Holding-set ::= SEQUENCE OF Book              --collection of books

Holding ::= SEQUENCE {
       b-num INTEGER,                         -- local key
       title VisibleString,
       author-name VisibleString,             -- last, first
       subj VisibleString OPTIONAL,
       type CHOICE {
              book SEQUENCE {
                     binding ENUMERATED {
                            hardcover(1),
                            paperback(2) },
                     num-pgs INTEGER },
              music SEQUENCE {
                     medium ENUMERATED {
                            record(1),
                            cd(2),
                            tape(3) },
                     length INTEGER },          -- in minutes
              movie SEQUENCE {
                     format ENUMERATED {
                            beta(1),
                            vhs(2),
                            reel(3) },
                     length INTEGER }},         -- in minutes
       language VisibleString DEFAULT "English",
       lc-num SEQUENCE {
              c-letter VisibleString,           -- one or more CAPS
              f-digit VisibleString,            -- one or more digits
              s-digit VisibleString OPTIONAL,   -- one or more digits
              cuttering VisibleString },        -- auth cutter number
       publisher-name VisibleString,
       publisher-addr VisibleString,            -- num, str, city, st
       checked-out BOOLEAN,                     -- TRUE if in library
       cost INTEGER }                           -- cost(whole dollars)
```

Figure 5. ASN.1 Specification for Database One

The database contains a group of holdings known as a *Holding-set*. Each *holding* consists of a unique identifier called the *b-num*. The *b-num* uniquely identifies each holding and is similar to the primary key of a relational database. Most of the remaining fields are self-explanatory. The *title* represents the holding's title. *Author-name* is the author's last and first names in that order. *Subj* is an OPTIONAL field containing the subject of the holding.

Type is a CHOICE field between three types of holdings: *book*, *music*, and *movie*. This means that the value for Type will depend on which of the choices is selected. Each choice is defined separately. If the value of the *type* choice is *book*, *type* will be defined as an ENUMERATED *binding* of either *hardcover* or *paperback*, and *num-pgs*, the number of pages in the book. If the value of the choice is *music*, *type* will be defined as an ENUMERATED *medium* of either *record*, *cd*, or *tape*, and *length*, the length of the music holding in minutes. Finally, if the value of the choice is *movie*, type will be defined as an ENUMERATED *format* of either *beta*, *vhs*, or *reel*, and *length*, the length of the movie.

Language is the language in which the holding was published. *Lc-num* is the Library of Congress number and is defined as a SEQUENCE of *c-letter*, *f-digit*, OPTIONAL *s-digit*, and *cuttering*. These are alphanumeric or numeric fields which compose a standard Library of Congress holding number of the form:

```
U260
B43
```

The *publisher-name* is the name of the holding's publisher. The *publisher-addr* is the number, street, city, and state of the publisher. *Checked-out* is a BOOLEAN value

which is TRUE if the holding is currently in the library, FALSE if the holding is checked out of the library. Finally, *cost* is the original cost of the holding in whole dollars.

A sample printfile conforming to the above specification is given in Figure 6.

```
Holding ::= {
  b-num 10 ,
  title "Joint Military Operations: A Short
History" ,
  author-name "Beaumont, Roger A." ,
  subj "Military Science" ,
  type
    book {
      binding hardcover ,
      num-pgs 245 } ,
  language "English" ,
  lc-num {
    c-letter "U" ,
    f-digit "260" ,
    cuttering "B43" } ,
  publisher-name "Greenwood Press" ,
  publisher-addr "Westport, Connecticut" ,
  checked-out TRUE ,
  cost 60 }
```

Figure 6. Sample ASN.1 Printfile for Database One

2. Database 2--Engineering Library

The second sample database is very similar to the first in that it contains information about a library's holdings. The ASN.1 specification for this database is given in Figure 7.

```
--****************************************
--
--  CDB-2 data definitions
--  Gino Celia, 1994
--
--****************************************

Component-two-module DEFINITIONS ::=
BEGIN

Item-set ::= SEQUENCE OF Item       -- collection of items

Item ::= SEQUENCE {
   i-num INTEGER,                   -- local key
   i-title VisibleString,
   a-name SEQUENCE {
      last VisibleString,
      first VisibleString,
      middle VisibleString OPTIONAL },
   subject VisibleString,
   type ENUMERATED {
      book(1),
      movie(2) },
   c-letter VisibleString,          -- one or more CAP LTRS
   f-digit VisibleString,           -- one or more digits
   s-digit VisibleString OPTIONAL,  -- one or more digits
   cuttering VisibleString,         -- author cutter number
   publisher SEQUENCE {
      p-name VisibleString,
      str-num VisibleString,
      str-name VisibleString,
      city VisibleString,
      state VisibleString,
      zip VisibleString },
   cost REAL,                       -- price in dollars and cents
   checked-out BOOLEAN }            -- true if checked out

END
```

Figure 7. ASN.1 specification for Database Two

In this case it is the Engineering Library and the holdings are referred to as items. Similar to Database 1, an *Item-set* is a group of individual *items*. *I-num* is the unique identifier of each item in the database. *I-title* is the item title. *A-name* the author name and is defined as a SEQUENCE of the *last*, *first* and OPTIONAL *middle* names. *Subject* is the item's subject. *Type* is the item type and is an ENUMERATION of either *book* or *movie*. *C-letter*, *f-digit*, OPTIONAL *s-digit*, and *cuttering* are all fields which represent the different portions of the Library of Congress number. The *publisher* field contains information about the item's publisher in six sub-fields. *P-name* is the publisher's name. *Str-num*, *str-name*, *city*, *state*, and *zip* are the publisher's street number, street name, city, state, and zip code, respectively. *Cost* is the original purchase price in dollars and cents, and *checked-out* is a BOOLEAN value which is TRUE if the book is checked-out and FALSE if the book is in the library.

A sample printfile conforming to the above specification is given in Figure 8.

```
Item ::= {
  i-num 21 ,
  title "A Breakfast for Bonaparte" ,
  a-name {
    last "Rostow" ,
    first "Eugene" ,
    middle "V" } ,
  subject "History" ,
  type book ,
  c-letter "E" ,
  f-digit "183" ,
  s-digit "7" ,
  cuttering "R749" ,
  publisher {
    p-name "National Defense University Press" ,
    str-num "1600" ,
    str-name "Pennsylvania Ave." ,
    city "Washington" ,
    state "DC" ,
    zip "20319" } ,
  value { 4199, 10, -2 } ,
  checked-out FALSE }
```

Figure 8. Sample ASN.1 Printfile for Database Two

Once the ASN.1 specifications have been developed along with conforming printfiles, a graphical tree representation can be created which shows the structure of the database pictorially. In Figures 9 and 10, tree depiction's for one item in each of the sample databases are presented using the graphical representations given in Chapter II.

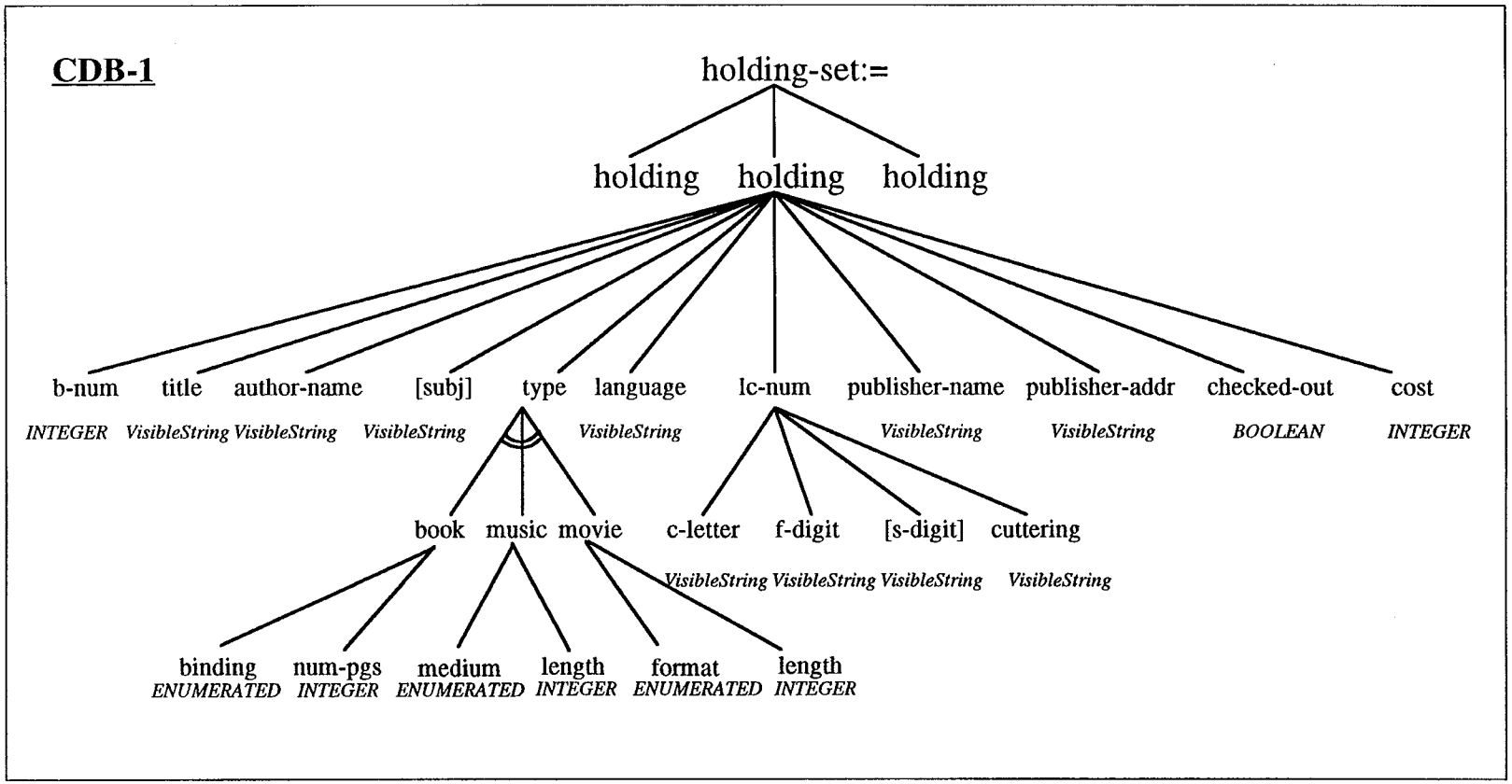


Figure 9. Graphical Representation of Database One

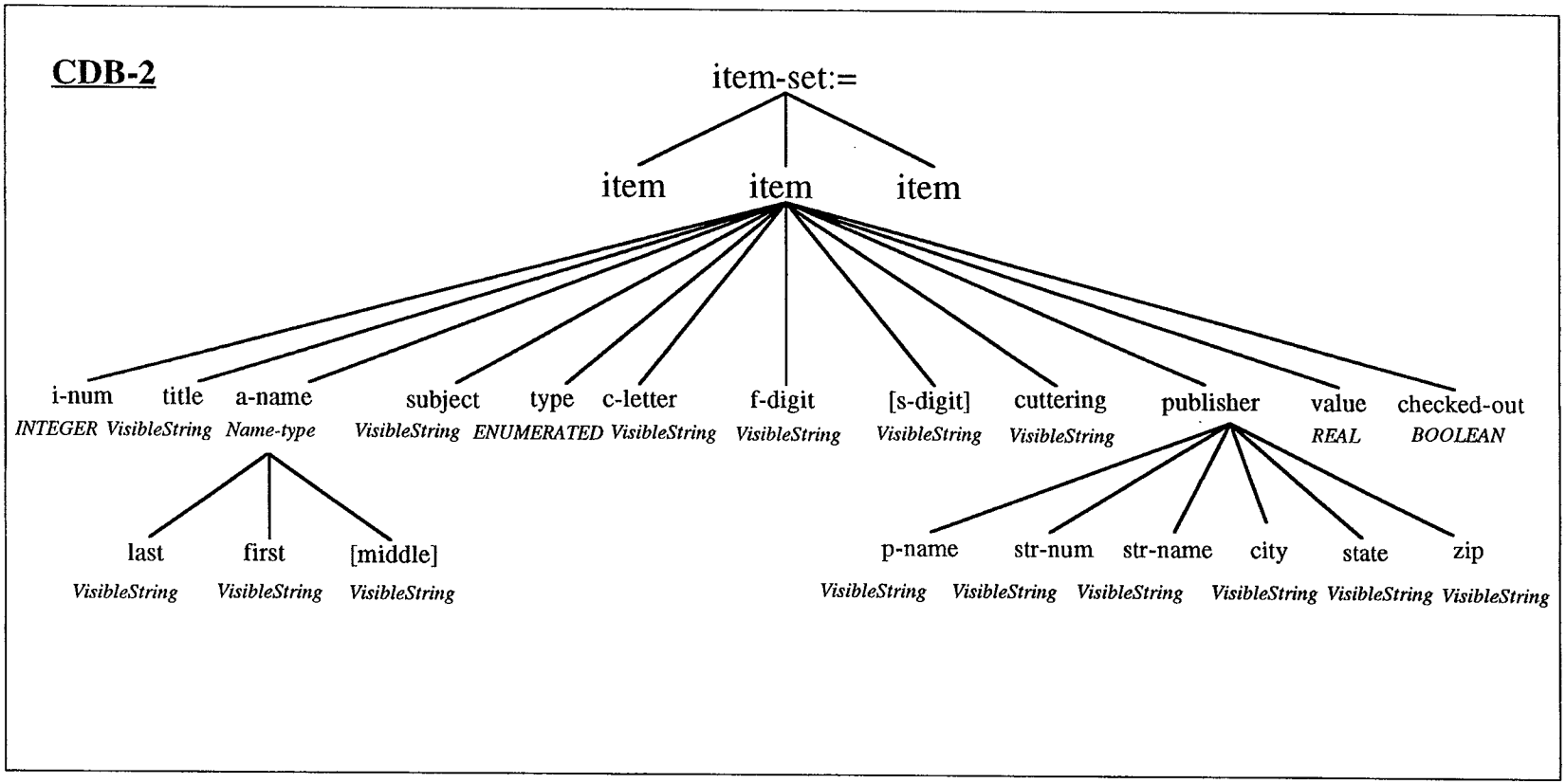


Figure 10. Graphical Representation of Database Two

The next chapter introduces the framework for classification of the semantic conflicts contained in these two sample databases. Many of these conflicts were contrived to better illustrate the framework and resolution strategies presented later in the thesis, but most integration attempts will contain several of the conflicts given in the framework. Throughout the thesis, we will utilize these sample databases to provide examples and test the implementation for correctness and completeness.

IV. FRAMEWORK OF SEMANTIC CONFLICTS IN ASN.1 DATABASES

The goal of this thesis is to develop a tool for assisting users in resolving the semantic conflicts which almost certainly arise when attempting to integrate two or more ASN.1-described databases. To accomplish this goal, we follow a three step approach.

The first step is to determine the scope and functionality of the end-system. This functionality is determined by the type of integration desired. Second, all possible schematic and data conflicts in ASN.1 databases must be identified and classified into a framework. Once this framework is established, the third step is to develop, test, and implement an algorithm for the resolution of all conflicts identified in the framework. This chapter addresses the first two steps of our approach. It discusses the two approaches of integration: tightly vs. loosely coupled. It then presents a classification scheme for organizing the types of conflicts into a logical framework. Chapters V and VI address the conflict resolution strategy and the implementation of the tool.

A. TYPES OF INTEGRATION

The approach supports two modes of integration: a loosely-coupled mode, and a tightly-coupled mode (Sheth and Larson, 1990). The modes differ in their use of the data dictionary and the presentations to the user as well as in the prerequisite knowledge requirements of the user regarding the schema of the component databases.

In the *loosely-coupled* mode, the dictionary is used merely as a look-up device for data descriptions. A set of common operators is presented to the users that include: 1) all supplied software tools, 2) all supplied canned database transactions, and 3) special interoperability operators, directly based on the ASN.1 DML, which allow the described I/O files to be queried, merged, and manipulated. No global schema integrating all the stored descriptions is attempted. The dictionary in this mode of operation may contain conflicts, such as synonyms, homonyms, and structural conflicts. The tool developed for this thesis helps the user in identifying and resolving these conflicts through a set of commands invoked in a graphical environment. The user bears the responsibility of understanding the semantics of each data file he wishes to use and resolving the conflicts that occur. The system hides all the networking details and provides a uniform set of operators for interoperability.

In the tightly-coupled mode, the dictionary assumes a more powerful role than simply being a lookup device—it acts as a global schema. The global schema is defined using the markup language as the Data Definition Language (DDL) and represents the integration of one or more component databases. To accomplish this integration, all semantic conflicts and inconsistencies must be resolved, and appropriate mappings to/from component databases should be defined. While the tool developed for this thesis does not perform the mapping for the federated schema, it supports conflict identification and resolution required as a prerequisite for this mapping. Again, the user is supplied with a uniform interface that will present a unified view of the databases, the software tools, and the

interoperability operators. The tightly-coupled approach requires more maintenance and attention from the users than the loosely-coupled approach, but offers greater consistency and demands less prior user knowledge about the structure of the underlying databases.

B. CLASSIFICATION OF SEMANTIC CONFLICTS

Regardless of which method of utilization is chosen, semantic conflict identification and resolution is a crucial step for facilitating the integration of databases. Semantic conflict in ASN.1 databases can be classified into two broad categories: schema conflicts and data conflicts (Kim and Seo, 1991). Schema conflicts are conflicts that occur at the level of the conceptual organization and definition of the database, while data conflicts occur as a result of differences in the actual data values returned from the different component databases. For our classification scheme, we utilized a model similar to that presented by Kim and Seo (1991). The anticipated schema and data conflicts are summarized in Figure 11, and addressed in detail in the remainder of this chapter. The semantic conflicts identified in this chapter apply to both tightly and loosely coupled integration approaches.

Schematic Conflicts

Name Conflicts

Synonyms -- same object named differently in different databases

Homonyms -- different objects named the same in different databases

Type Conflicts -- same objects have different types in different databases

Structural Conflicts

Grouping Conflicts -- horizontal or vertical grouping differences in different databases

Sequence Conflicts -- sequences defined differently in different databases

Optional Item Conflicts -- optionality defined differently in different databases

Choice Conflicts -- choices defined differently in different databases

Data Conflicts

Precision Conflicts -- different precision utilized in different databases for the same object

Unit Conflicts -- different units utilized in different databases for the same object

Expression Conflicts -- different expressions utilized in different databases for the same object

Figure 11. Summary of ASN.1 Semantic Integration Conflicts

1. ASN.1 Schematic Conflicts

Conflicts which occur due to differences in the structure of the database are known as schematic conflicts. In ASN.1 tree structures, these conflicts can occur at the individual nodes of the ASN.1 trees, and are sometimes referred to as node conflicts. For our purposes, schematic node conflicts can be one of three basic types:

Name Conflicts

Synonyms—the same object is named differently in different component databases. (e.g., the title of each holding is referred to as "title" in CDB1 and "i-title" in CDB2.)

Homonyms—different objects are named the same in different component databases. (e.g., the value for "checked-out" has the same name, but different meanings in each database. A TRUE value for this field in CDB1 indicates that the item is in the library while it indicates the item is checked-out of the library in CDB2.)

Type Conflicts

Type Conflicts—the same object is defined using different base types in different component databases. (e.g., item "cost" is defined as an INTEGER [whole dollars] in CDB1 and a REAL [dollars and cents] in CDB2.)

Structural Conflicts

Grouping Conflicts—objects grouped differently either vertically, horizontally, or both in two different component databases. (e.g., "author-name" is defined as a VisibleString in CDB1, and the equivalent field "a-name" is defined as a SEQUENCE OF VisibleStrings in CDB2. This is a combination of both a vertical and horizontal grouping conflict since the VisibleStrings in CDB2 must be combined horizontally into a new field and then moved vertically one level up to be equivalent to the author-name field in CDB1.)

Sequence Conflicts--SEQUENCES defined differently in each component database. (e.g., a name field may be a SEQUENCE of first-name→last-name fields in one database, but last-name→first-name in the other.) Note that this is a potential conflict for SEQUENCE structures only, since SET structures are not ordered.

Optional Item Conflicts—an object is optional in one component database, but not the other. (e.g., "subj" is OPTIONAL in CDB1, but "subject" is mandatory in CDB2.)

Choice Conflicts—an object or group of objects is defined as a choice in one component database, but not the other. (e.g., "type" is a CHOICE in CDB1, but ENUMERATED in CDB2.)

2. ASN.1 Data Conflicts

Data conflicts occur when two like objects in different component databases are stored in compatible formats, but the data itself is incompatible or the data is incorrect. Data conflicts include:

Precision Conflicts—data for the same object in two different component databases are stored with different precision or granularity. (e.g., Item values are rounded to the nearest dollar in CDB1, but recorded as dollars and cents in CDB2)

Unit Conflicts—data for the same object in two different component databases are stored with different units, making their comparison incompatible. (e.g., Item values might be stored in US Dollars (US$) in one component database, but stored in Japanese Yen (¥) in the other.)

Expression Conflicts—similar data in two different component databases is represented by different expressions in each database. (e.g., Book names could be abbreviated in one component database, but not the other.)

Note that several other types of data conflicts such as missing or incorrect data can (and usually do) exist. Since these conflicts are not detectable through examination of the

individual component database schemas, the user has few options available to correct these conflicts at schema definition time. Even if these types of conflicts could be detected, the only way to resolve them is to add to or modify the data entries in the original databases themselves. There is no filter, algorithm, or operator available to resolve these types of conflicts at the virtual level during schema generation. For this reason, those conflicts which cannot be identified at schema generation time are not addressed in this thesis.

V. CONFLICT RESOLUTION

The purpose of this chapter is to discuss techniques for resolving the conflicts described in the previous chapter. These techniques are implemented using graphical commands that the user can utilize to resolve these conflicts. The chapter presents a technique for the resolution of each conflict previously identified, including a discussion of the tools presented to the user to implement these resolution methods.

Once the framework for conflict resolution is established, a means for resolving each conflict identified in the framework must be identified. Although the ideal system would include algorithms which automatically detect the conflicts identified in the previous chapter and resolve them heuristically, this is not likely to be feasible for several reasons. First, any automation of conflict resolution would require each component database to maintain strict standardized data dictionaries—a practice which is currently far from the norm. Additionally, some assumptions about the data to be merged must be made which would not necessarily apply universally to all sample databases. Future research, especially in artificial inteligence, may lead to further automation of the process. For this thesis we provide tools for the user to aid in schema/data adjustment and conflict resolution. Additionally, we assume the user or system integrator is knowledgable enough about the structrue and semantics of the component databases.

The success of the Data Manipulation Language (DML) is dependent on the removal of all possible semantic conflicts. The following sections discuss the means for the removal of these conflicts.

A. SCHEMA LEVEL RESOLUTION

Name Conflicts

Synonyms and Homonyms: Naming conflicts occur due to either synonym or homonym conflicts. In order to resolve this conflict, a command for renaming fields in the virtual schemas must be provided. This command can be used for synonyms to change a synonym node name in one database to match the node name of the equivalent field in the other component database. Similarly for homonyms, the command can be used to change one of the node names to a different, unique name.

Type Conflicts

Type conflicts occur when the like objects are defined using different base types. To resolve this type of conflict, a command must be provided to dynamically change the type of one of the objects. Since not all types are interchangable, a table of allowed conversions must be specified. For instance, almost all types can be converted to the VisibleString type, however, the reverse is not true. Alphanumeric characters which make up a VisibleString cannnot meaningfully be converted to INTEGERs or REALs. For our purposes, the allowed type conversions will conform to ANSI C type casting rules since the program is implemented in that language. These rules, as they apply to this thesis, are presented in detail in Chapter VI.

Structural Conflicts

Grouping conflicts: Vertical and horizontal grouping conflicts usually occur due to differing levels of detail or different information requirements in the component databases.

For horizontal grouping conflicts, two operations are required for successful conflict resolution: concatenate and subset. Concatenate allows two horizontal nodes at the same level to be combined into one. Subset allows one node to be divided into two new nodes at the same level in the ASN.1 tree diagram. The subset operator requires specification from the user in order to determine how to divide the node (e.g., dividing "author-name" in CDB1 into two separate nodes, "last-name" and "first-name").

Vertical grouping conflicts resolution also require two operations: collapse and expand. Collapse causes a child node to be merged with its parent into a single node. Expand divides a parent node into a new parent-child node combination. Like the horizontal subset operator, the expand operation requires the user to specify how the data is distributed between the two nodes to perform the node division.

Sequence Conflicts: Occur when SEQUENCES are defined differently in each database. To resolve this type of conflict, the user must be able to rearrange the items in a SEQUENCE strucutre in one component database so that they coincide with the defined SEQUENCE structure in the other database.

Optional Item Conflicts: Occur due to a difference in optionality. To resolve this type of conflict, the user must be given a command which removes the optionality of a given

node. If a node becomes mandatory and there is no data value for a given instance, a NULL value is inserted in the data item of that node.

Choice Conflicts: Occur when an object is defined as a CHOICE in one database, but not the other. To resolve this conflict, the user must have the ability to redefine the CHOICE node as a required one which matches the type of the corresponding node in the other component database.

B. DATA LEVEL RESOLUTION

Precision Conflicts: Occur when data in two different databases are stored with different precision or granularity. Resolution of this type of conflict requires transformation of the values of one of the data sets to the other's precision. For example, if books were rated on a scale of 1-10 in one database and A-F in the other, rating of the data items in one database would need to be mathmetically converted to the other's. In this case, a numeric value might be assigned for each of the A-F grades in order to allow a proper comparison between the two databases. The conversion process may require information from the designers of the databases.

Unit Conflicts: Occur when data in two different databases are stored with different units. This conflict, much like the precision conflict, requires transformation of one data set to the other's units. For instance, if the value of the books were stored in dollars in one database and yen in the other, the user would have to supply either a dollar-to-yen or yen-to-dollar conversion formula in order to unify the units in each database.

Expression conflicts: This is perhaps the most difficult data conflict to resolve, and in fact some expression conflicts may not be correctable at schema generation time. These conflicts occur when different expressions are used to store the same data object. If the expression conflict occurs consistently throughout the database, it may be correctable by mapping the data from one expression to another with the help of user input. However, if the expression conflict occurs randomly or sporadically across the data set, there is probably no easy algorithm for conflict resolution, and some data inconsistencies may remain.

VI. IMPLEMENTATION OF CONFLICT RESOLUTION

The culmination of the research presented in the previous chapters is a working prototype of a database integration system designed to aid in the resolution of schema and data conflicts resulting from that integration. This chapter discusses platform, language, and implementation decisions and serves as a user's guide to the software. A full source code listing is provided in the Appendix.

A. PLATFORM AND LANGUAGE CONSIDERATIONS

This software is primarily designed for end-users who access multiple databases on the internet or through other local and wide area networks. With that in mind, we chose to develop the prototype in the UNIX/X Window environment in order to support the majority of the target audience. While some end-users will undoubtedly access their data from other environments, it is generally believed that most users will utilize a UNIX workstation running X Windows.

X Windows is a platform-independent graphical environment originally designed for the UNIX operating system and now being ported to other operating systems. However, most graphical software built for X Windows is not designed and implemented at the lower levels of the X libraries. Instead, several vendors have developed toolkits of library routines designed to ease programming in X Windows and provide a uniform look-and-feel to software developed using these toolkits. Sun Microsystems'

OpenWindows™ and Open Software Foundation's Motif™ are two examples of such toolkits based on the X Windows routines. Since Sun has recently announced that it was discontinuing OpenWindows™ and bundling Motif™ with Solaris™, its version of the UNIX operating system, we decided to build the system in Motif™. Unfortunately, the existing computer resources at the Naval Postgraduate School do not include Sun workstations with Motif. However, the Silicon Graphics workstations available in the Visualization Lab at the school do include the Motif development libraries, and were therefore chosen as the hardware platform of choice to develop the prototype. Specifically, the prototype system was developed on Silicon Graphics workstations running IRIX 5.2 (SGI's UNIX) and Motif 1.2.3/X11R5.

B. IMPLEMENTATION DECISIONS AND TOOLS

The first step of the implementation was determining the software design and functionality. Since the final system will include additional functionality, the design allows for functionality which is not implemented in the current version. For example, the command window has an area dedicated for the DML commands, but no DML commands are implemented in this version. The user is provided with a multi-window environment consisting of a window for each ASN.1 tree and a central control window which provides the conflict-resolution commands. The user loads two or more graphical ASN.1 database specifications/printfile pairs in separate windows, identifies and resolves the conflicts, and then performs DML functions to manipulate the data of the various databases.

It is important to note that any changes the user makes to any ASN.1 specification or printfile are virtual and do not affect the original files. The program automatically generates output files which contain the new trees with semantic conflicts resolved. Later, the user can load the updated specifications and printfiles and thus skip the conflict resolution step and start utilizing the DML immediately.

1. NCBI Toolkit

The implementation of the prototype was aided greatly by the use of a series of C programs and libraries called the NCBI Toolkit which was developed at the National Center for Biotechnical Information (NCBI, 1993). These libraries include extensive routines for the handling and manipulation of ASN.1 encoded specifications and printfiles. The complete software toolkit needs to be installed on the end user's system in order to perform the initial parsing of the ASN.1 specifications and printfiles into a format readable by a tree generation program that displays them in a graphical format in preparation for conflict identification and resolution. The toolkit is available through anonymous FTP to *ncbi.nlm.nih.gov*.

2. OSF/Motif™

Motif is a standard user interface toolkit developed and supported by the Open Software Foundation (OSF) and its member companies. Motif includes the Motif widget set, which is based on the Xt Intrinsics and include graphical user interface components such as buttons, sliders, menus, etc. Motif has a distinctive three-dimensional beveled appearance which is described more fully by the Motif Style Guide. While Motif is not

available for free in the public domain, it is bundled with many major operating systems. Figure 12 shows a typical architecture of a Motif/Xt application (Young, 1994).

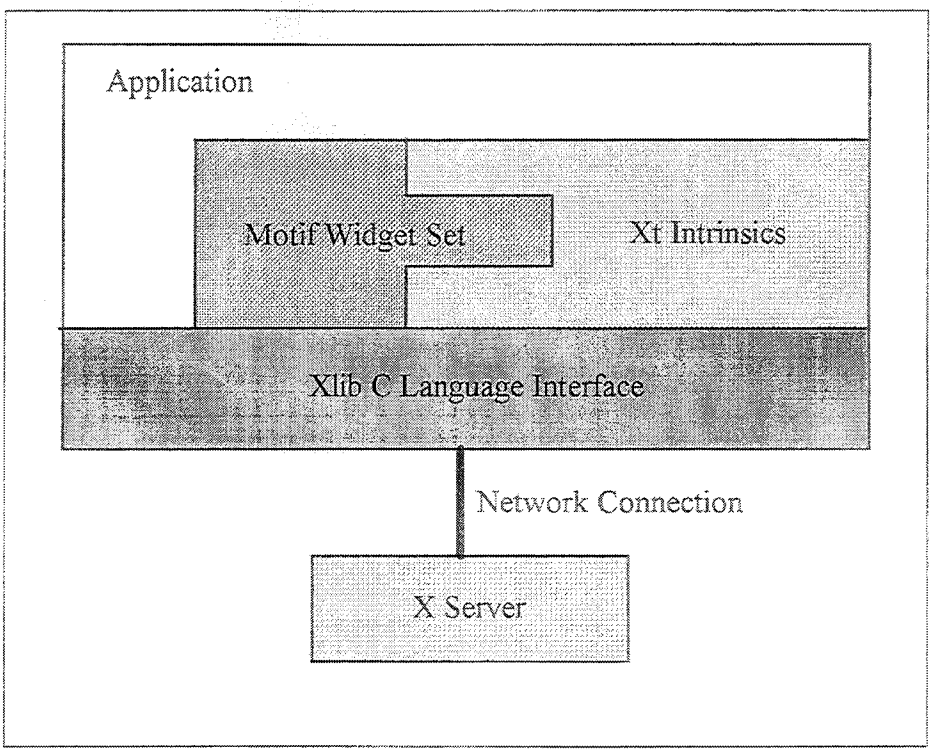


Figure 12. Architecture of a Motif/Xt Application

The following sections describes the procedures for utilizing the software and provides samples of the screen output at various points in the execution. There is currently no on-line or context-sensitive help provided by the system.

C. PROTOTYPE DESIGN

1. Data Flow

Before discussing the actual operation of the program, it is important to understand the flow of data in the system. Operation of the application actually requires the use of two programs. Refer to Figure 13 during the discussion of data flow.

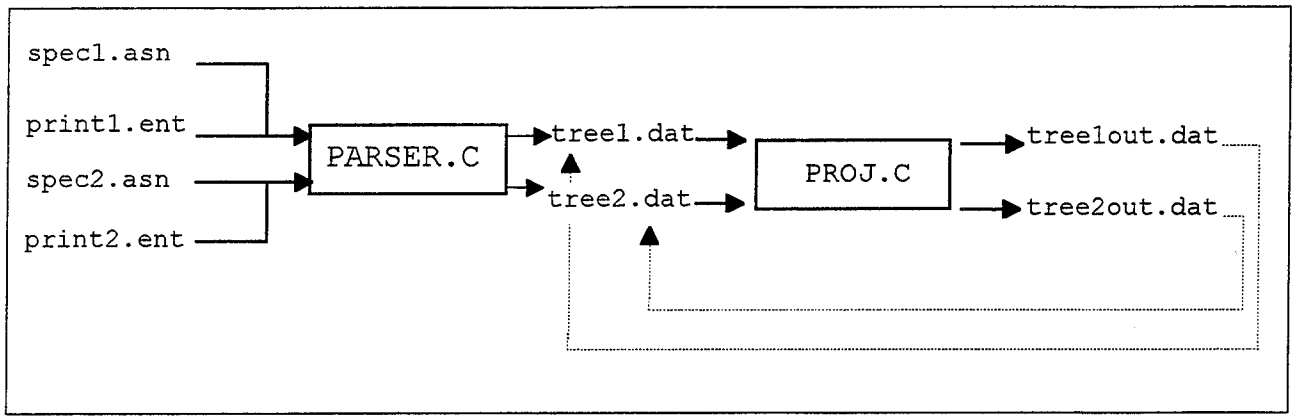


Figure 13. Data Flow Model

As the figure shows, processing ASN.1 databases is a two-stage procedure. First, the ASN.1 specification and its associated printfile are processed through the `parser.c` program. This utility is a modification of routines in the NCBI toolkit which validates the specification, then checks the printfile against the valid specification and finally outputs an ASCII text file for use in the second stage of the application. This text file consists of sets of parent-child node numbers and labels formatted for use in the `proj.c` program. This program makes use of a Motif-based tree widget written by Douglas Young of SGI (1994). After manipulation of the trees by this program, new ASCII output files are automatically generated with conflicts resolved for future use. There is currently no facility to parse the ASCII files back into ASN.1 specifications and printfiles; this task is left for future research.

2. Main Programs

Following is a list of all the program files associated with the implementation:

`parser`	-- executable parser program
`parser.c`	-- source code for parser

`ginoprint.c`	-- replacement of NCBI asnprint.c routines (produces ASCII tree files)
`proj`	-- executable integration program
`proj.h`	-- header file for proj
`proj.c`	-- source code for proj
`Tree`	-- Motif resource file for proj
`TreeP.h`	-- public header file for tree widget
`Tree.h`	-- private header file for tree widget
`Tree.c`	-- tree widget
`*.asn`	-- an ASN.1 specification
`*.ent`	-- an ASN.1 printfile
`*.out`	-- an ASN.1 specification or printfile which has been processed by the parser (validated and output)
`*.dat`	-- ASCII tree file

3. Initial Screen

Figure 14 shows the initial screen of the prototype. It consists of a command window at the top that contains the conflict resolution commands and the DML commands, and two or more database windows that display the graphical ASN.1 databases to be manipulated. The user should ensure that the proper tree is selected using the **Active Tree** radio buttons before selecting any **Resolution Commands**. The **Record** arrow buttons select the *first*, *previous*, *next*, and *last* record of the active tree, respectively.

4. Using the Prototype

The conflicts presented in the previous two chapters will now be discussed in the context of the implementation. Each type of conflict will again be presented and the solution given in terms of program operation. For each conflict, an example dialog box and the resulting ASN.1 tree diagram will be given. All examples assume the initial graphical ASN.1 trees given in Figure 14.

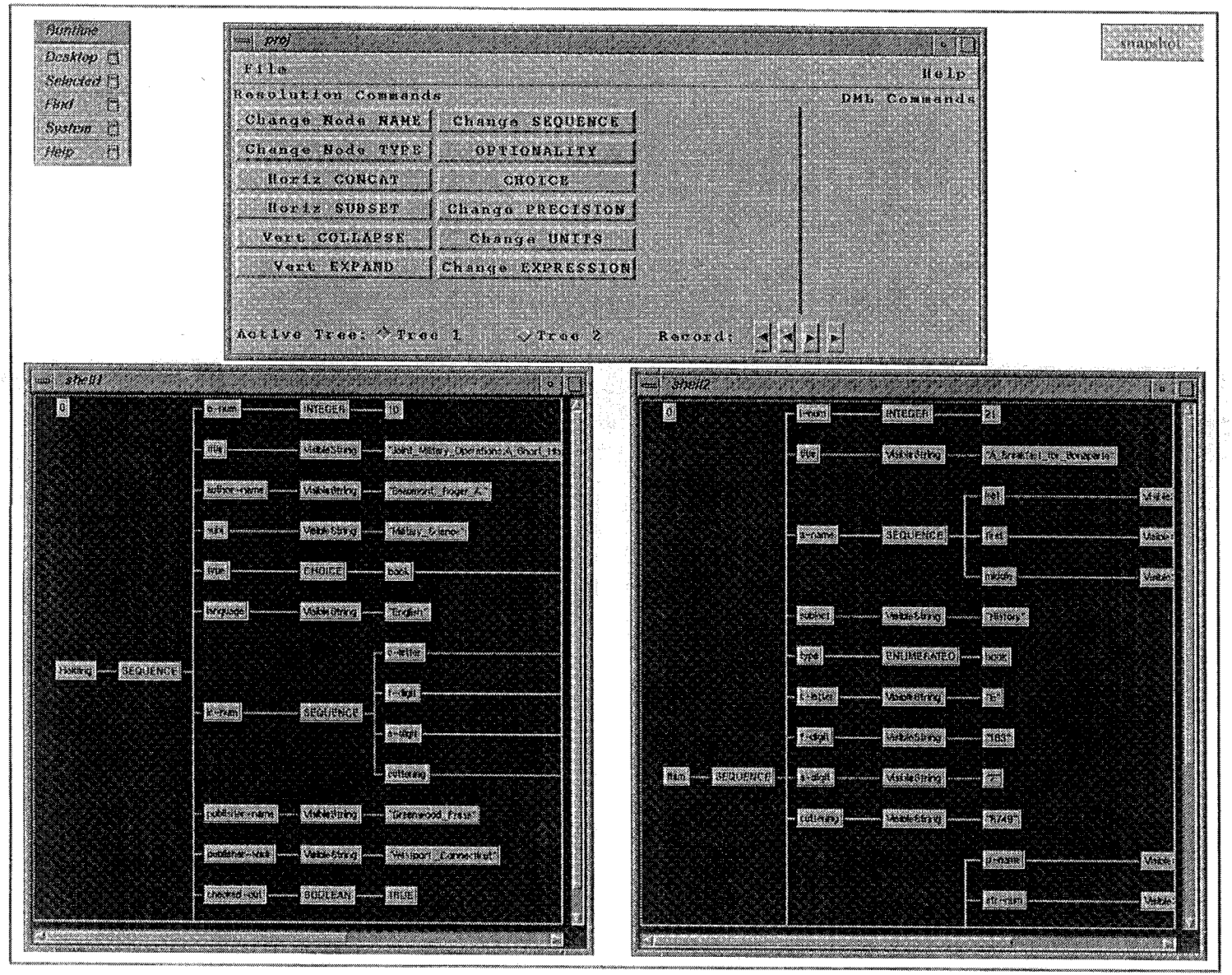


Figure 14. Initial Screen

SCHEMA LEVEL RESOLUTION

Name Conflicts

Synonyms and Homonyms: Naming conflicts occur due to either synonym or homonym conflicts. To correct these types of conflicts, the user selects the **Change Node NAME** button. The dialog box shown in Figure 15 is displayed. The user enters the name of the node to be changed and a new name.

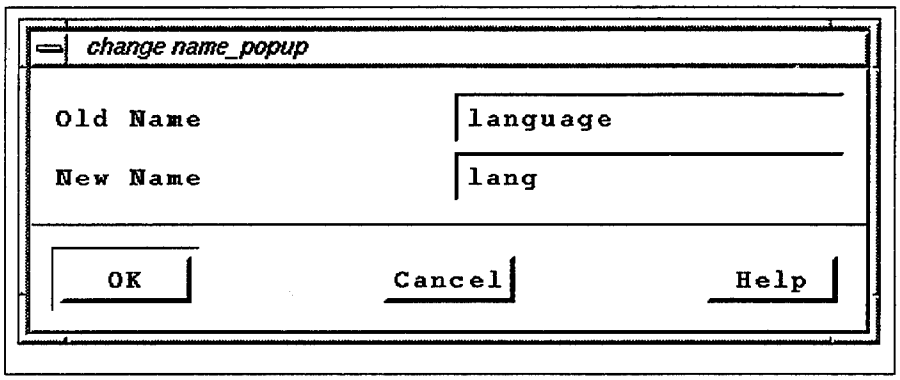


Figure 15. Change Node NAME Dialog

After pressing OK, the node name is changed for each record in the active tree as shown in Figure 16.

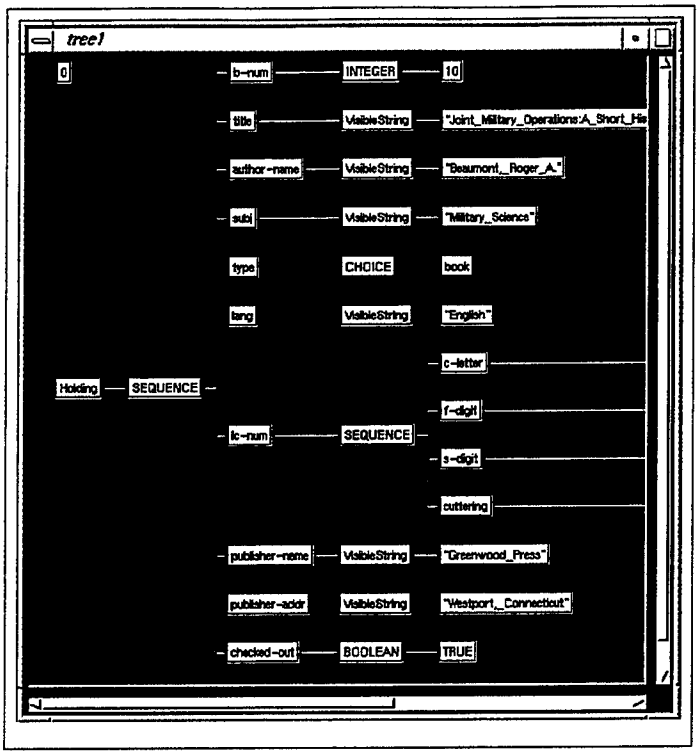


Figure 16. Tree Diagram with New Node Name

Type Conflicts

Type conflicts occur when the like objects are defined using different base types. To correct type conflicts, the user selects the **Change Node TYPE** button. The dialog box given in Figure 17 is displayed. The user then enters the node name whose type is to be changed and a new type, and presses OK.

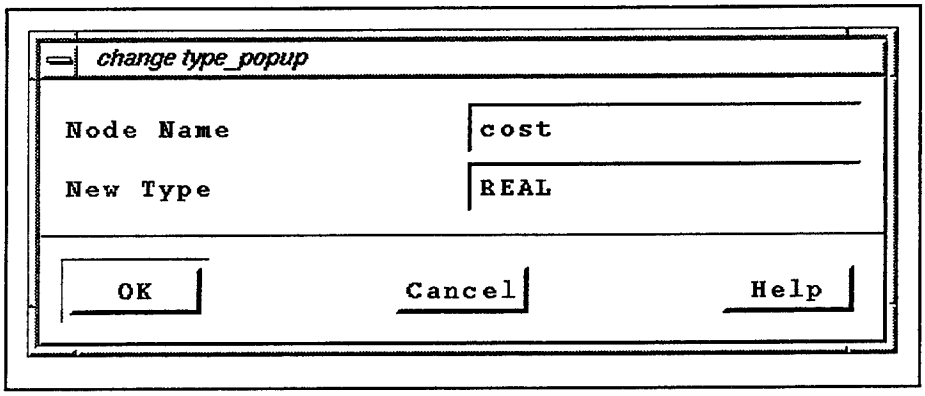


Figure 17. Change Node TYPE Dialog

If the user attempts a type change which is not allowed, an error message appears and the type is not changed. Legal type changes are those which conform to normal C typecasting rules. If the type change is legal, the type of the selected node is changed as shown in Figure 18.

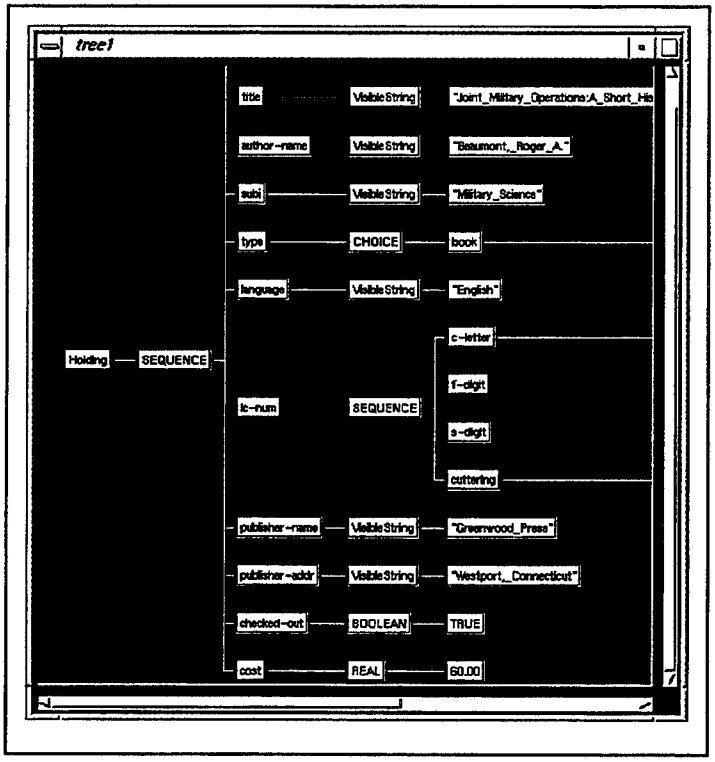


Figure 18. Tree Diagram with New Node Type

Structural Conflicts

Grouping conflicts: Vertical and horizontal grouping conflicts usually occur due to differing levels of detail or different information requirements in the component databases.

For horizontal grouping conflicts, two operations are required for successful conflict resolution: concatenate and subset. To perform a horizontal concatenation, the user clicks the **Horiz CONCAT** button. The dialog shown in Figure 19 is displayed.

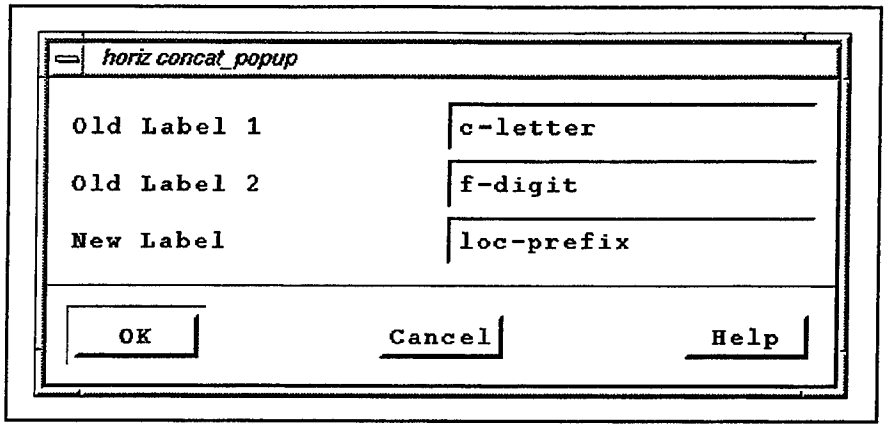


Figure 19. Horizontal CONCAT Dialog

After entering the names of the two nodes to merge together, a new label, and clicking OK, the resultant tree diagram given in Figure 20 is produced.

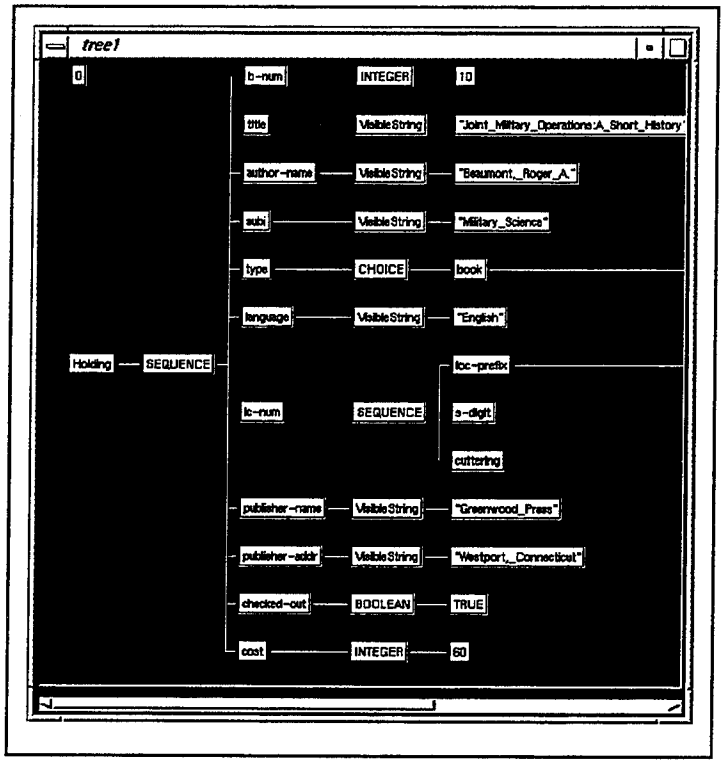


Figure 20. Tree Diagram After Horiz CONCAT

Horizontal subsets are produced in much the same manner. By selecting the **Horiz SUBSET** button, a dialog is generated which asks for the node to divide and the character separating the data to be divided. This dialog is given in Figure 21.

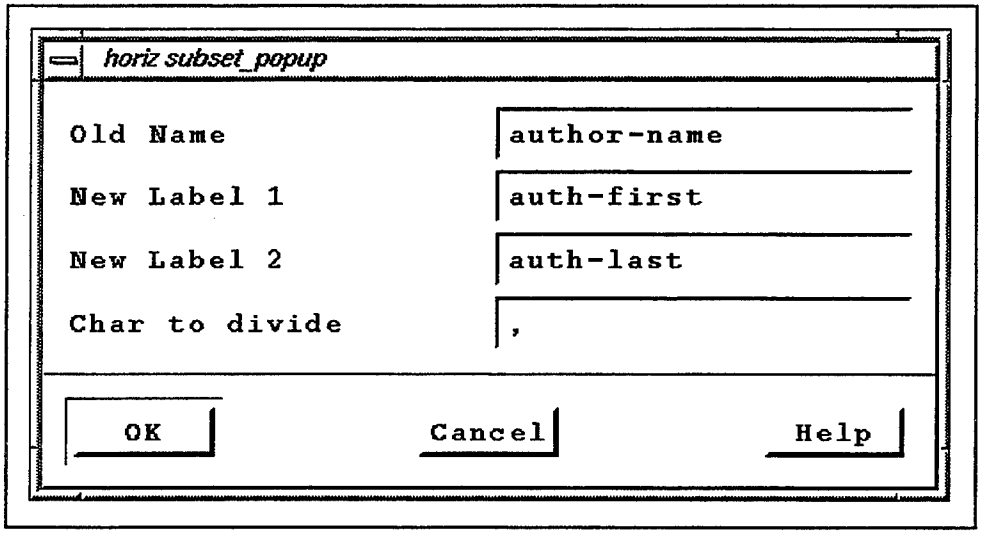


Figure 21. Horiz SUBSET Dialog

A tree diagram with a horizontal subset is shown in Figure 22.

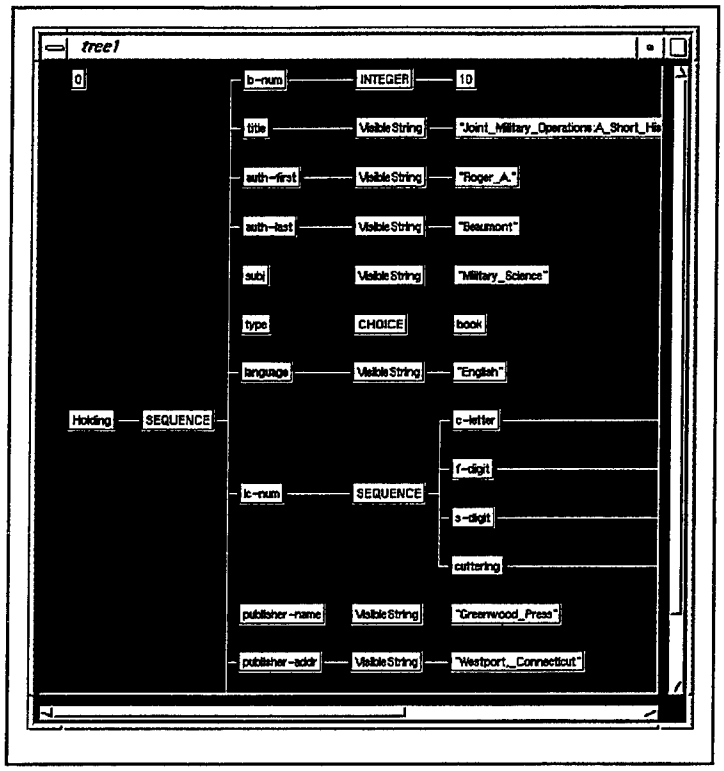


Figure 22. Tree Diagram After Horiz SUBSET

Vertical grouping conflicts are resolved with the **Vert COLLAPSE/Vert EXPAND** buttons, whose operation is identical to the Horiz CONCAT and SUBSET buttons.

Sequence Conflicts: Occur when SEQUENCES are defined differently in each database. To resolve this type of conflict, the user selects the **Change SEQUENCE** button. The dialog box shown in Figure 23 is displayed.

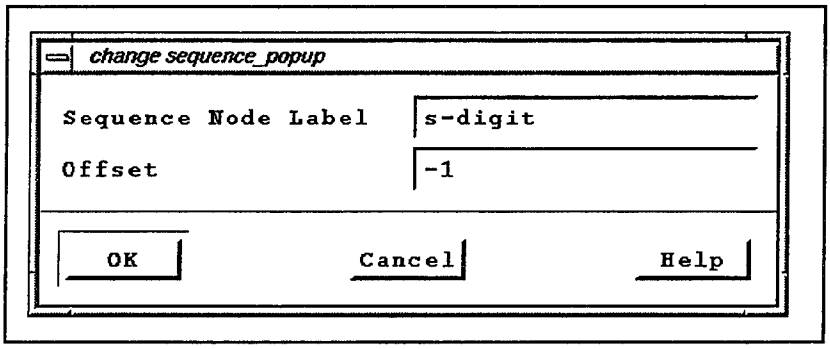


Figure 23. Change SEQUENCE Dialog

The user then enters a node label from a SEQUENCE in the active tree, and an offset for the number of places to move the node. A negative offset moves the node up, a positive offset moves it down. For example, the entries in Figure 23 will move the node labeled *s-digit* UP 1 place. Figure 24 shows the modified tree after executing the above command.

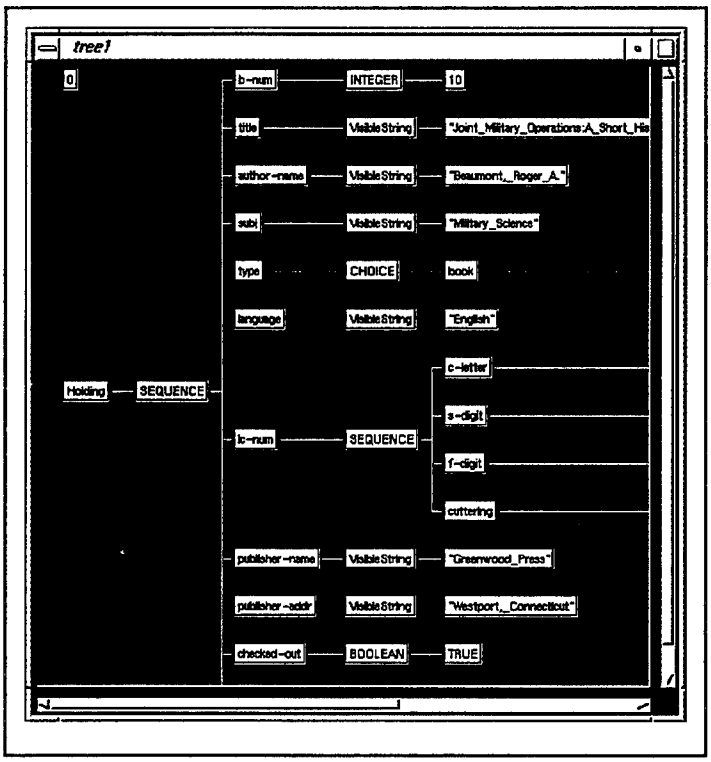


Figure 24. Tree Diagram After Change SEQUENCE

Optional Item Conflicts: Occur due to a difference in optionality. When the **OPTIONALITY** button is selected, the user is prompted for the label of an optional node in the active tree, as shown in Figure 25.

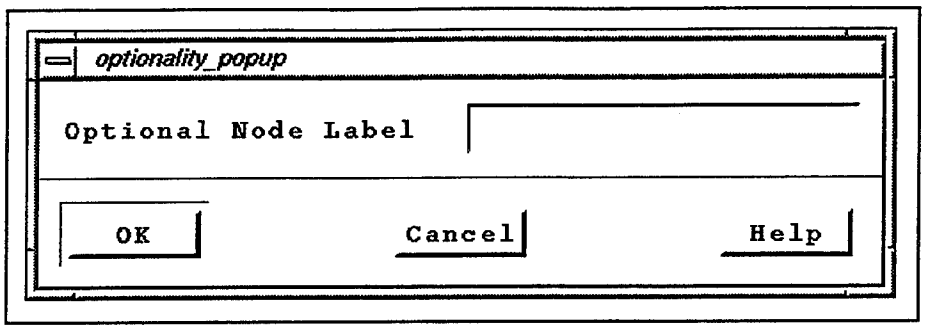


Figure 25. Optionality Dialog

If a valid node label is entered, the optionality of that node is removed. Any instance of the tree for which the optional node was not included is automatically filled with a NULL value.

Choice Conflicts: Occur due to a difference in choice definitions. Like the optionality operation above, selecting the **CHOICE** button prompts the user to enter the label of a choice node as shown in Figure 26. If a valid node label is entered, the CHOICE node is transformed into a SEQUENCE node whose children correspond to the different choice values. Each item in the sequence will have a NULL data value except the original choice node.

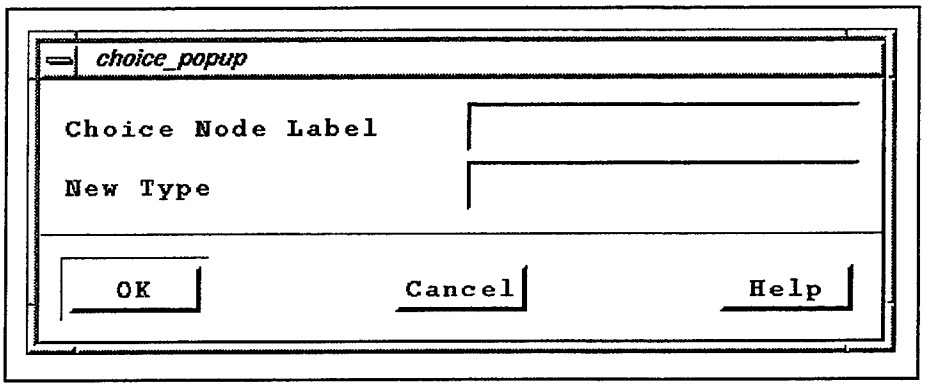


Figure 26. Choice Dialog

B. DATA LEVEL RESOLUTION

Precision Conflicts: Occur when data in two different databases are stored with different precision or granularity. In order to correct this conflict, the user selects the **Change PRECISION** button. He is then prompted for a node label from each tree. The program automatically maps the values from Tree 2 to corresponding values in Tree 1.

Unit Conflicts: Occur when data in two different databases are stored with different units. To correct this conflict the user selects the **Change UNITS** button. After entering a node label from the active tree, the user enters a multiplier value in the dialog box shown in Figure 27.

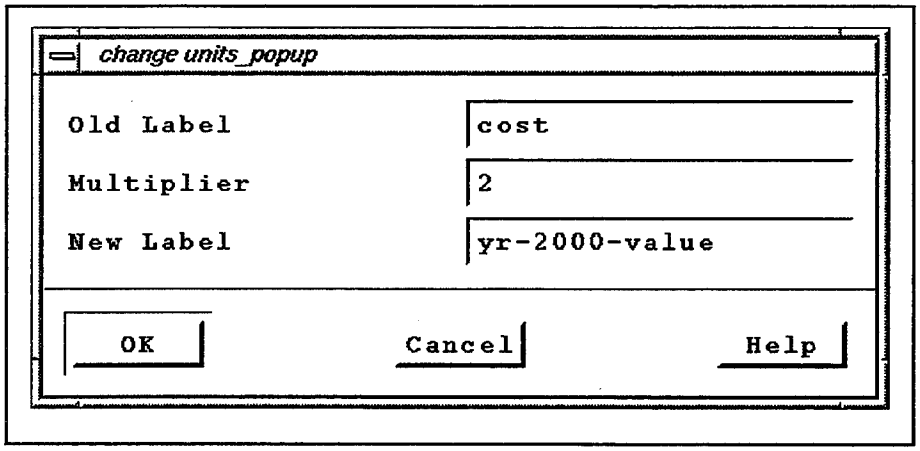


Figure 27. Change UNITS Dialog

All data values for the selected node are multiplied by this value to scale them to the new unit, resulting in the tree diagram given in Figure 28.

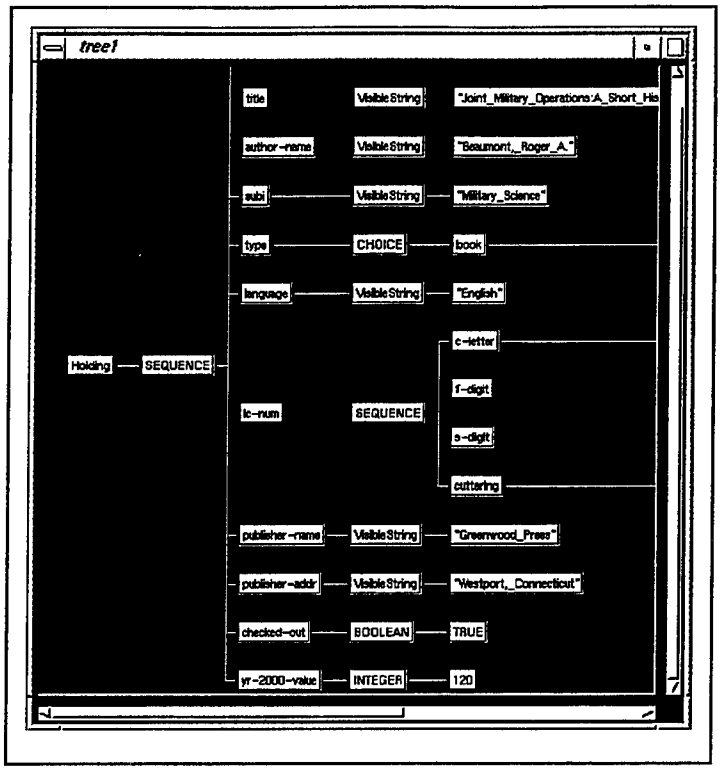


Figure 28. Tree Diagram After Change UNITS

Expression conflicts: These conflicts occur when different expressions are used to store the same data object. The **Change EXPRESSION** button is the only button which allows a user to actually manipulate a single data value for a single instance only. When the user clicks this button, he is prompted for the old and new data values with the dialog box shown in Figure 29.

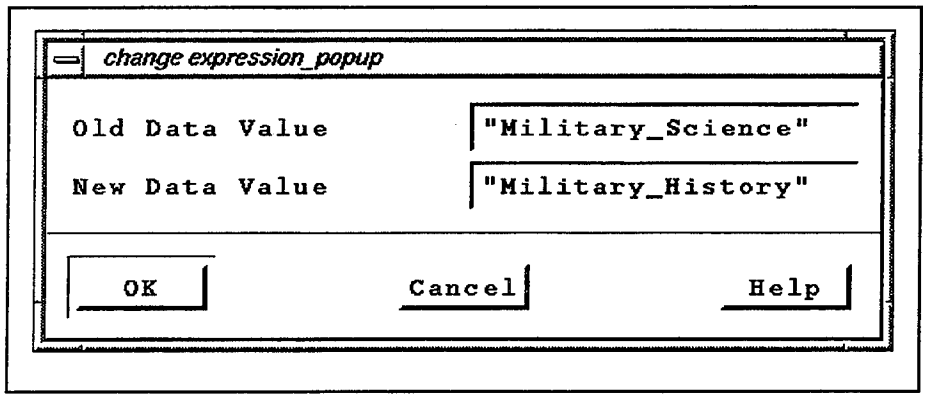


Figure 29. Change EXPRESSION Dialog

Figure 30 shows the tree diagram after the data value has been modified.

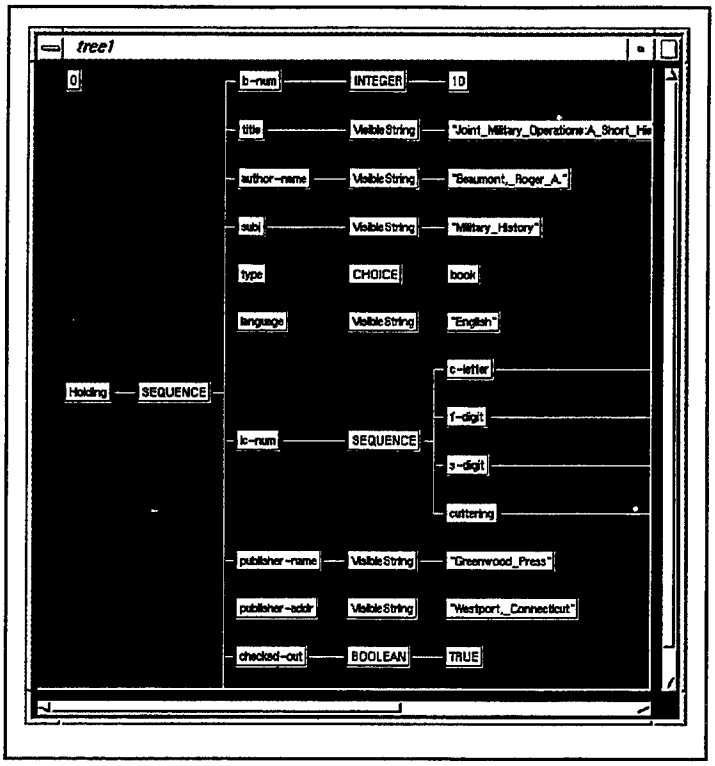


Figure 30. Tree Diagram After Change EXPRESSION

VII. CONCLUSION

In this effort, we have attempted to show how ASN.1 can be successfully utilized as both a Data Definition and Data Manipulation Language for the integration of heterogeneous databases. This successful integration depends on the resolution of all possible schema and data conflicts before the user manipulates the databases using an appropriate DML. While others have studied these conflicts as they apply to more conventional database styles like the relational and object-oriented models, we have presented a classification and resolution strategy designed around the unique requirements of ASN.1-described databases and documents. The resolution of these conflicts will hopefully allow users to access and integrate data in databases having heterogeneous formats in a uniform manner utilizing an easy to use graphical user environment.

The ideas addressed in this thesis are particularly applicable to issues of the Department of Defense and the Department of the Navy. As with most modern large organizations, DoD and DoN are discovering a myriad of incompatible stovepiped databases that evolved over a period of time without careful guidance or planning. This situation resulted in data duplication, data inconsistencies, inflexibility, limited data sharing, and maintenance problems. As part of the DoD Corporate Information Management (CIM) Initiative, examination of various database integration strategies and the development of data warehouses is being intensively pursued. Although ASN.1

integration has not yet encountered widespread use, future research using ASN.1 or other markup languages, like SGML, should yield workable systems which may indeed play a role in the Department of Defense's progression toward a uniform, federated database environment.

A. LESSONS LEARNED

Although this thesis originally intended to produce a full-scale application which implemented all the aspects of the classification and resolution schemes presented herein, the program had to be scaled down to a working prototype with the look-and-feel of the full system, but with limited functionality. Unfortunately, it is difficult to accurately predict the amount of time required to produce a piece of software and in this case, some aspects of the system will require follow-on work for its completion. Some of this shortcoming is explained by the author's limited programming experience in the C language, the dependence on source code libraries obtained from external sources which required a significant amount of time for familiarization.

B. FUTURE WORK

While this thesis serves as good start toward the development of a full-scale conflict resolution and database integration system, some areas require future research. First, this thesis focuses on the loosely-coupled approach to data integration. The tightly-coupled approach discussed in Chapter IV simplifies the user interaction with component databases by providing a single unified schema, but is much more difficult to implement due to the

generation of a federated schema. However, this approach should be included in the final product.

Additionally, there has been demand throughout the distributed database community for integration tools which act on the generated output reports of the component databases rather than the local schemas and data. Integration of these reports would allow the local database administrators to change various aspects of their individual schemas without affecting the integration as long as the output reports remained constant. Since this is usually the case, development of a system which allows output report integration is the logical answer to this issue. Future thesis efforts should look specifically in this type of integration.

APPENDIX
PROGRAM LISTING

PARSER.C

```
/* Thesis project
*
*  Parser which converts ASN.1 specs and printfiles to
*  ASCII tree files
*
*  by: LT Gino Celia, Jr., USN
*
*/

#include <asnbuild.h>

#define NUMARGS 4
Args myargs[NUMARGS] = {
   { "Input spec", "spec1.asn", "Book", NULL, FALSE, 'i', ARG_FILE_IN,
0.0, 0, NULL},
   { "Output file", "result.out", NULL, NULL, FALSE, 'o', ARG_FILE_OUT, 0.0,
0, NULL},
   { "Output data", "strm.out", "Book", NULL, FALSE, 'd', ARG_DATA_OUT, 0.0,
0, NULL},
   { "Input data", "print1.ent", "Book", NULL, FALSE, 'p', ARG_FILE_IN,
0.0, 0, NULL},};

extern void AsnTxtReadValFile PROTO((AsnModulePtr amp, AsnIoPtr aip, AsnIoPtr
aipout /* ,
AsnIoPtr encode */));

Int2 Main ()

{
   AsnIoPtr       aip = NULL,
             aipprint = NULL,
             aipout = NULL;
   AsnModulePtr amp = NULL;
   AsnTypePtr     atp;
   FILE     *fp;
   DataVal  value;

/*
   GetArgs("Parser 1.0", NUMARGS, myargs);

   aip = AsnIoOpen(myargs[0].strvalue, "r");
   aipprint = AsnIoOpen(myargs[3].strvalue, "r");
   aipout = AsnIoOpen(myargs[2].strvalue, "w");
   fp = FileOpen(myargs[1].strvalue, "w");
*/
   aip = AsnIoOpen("spec1.asn", "r");
```

```
    aipprint = AsnIoOpen("print1.ent", "r");
    aipout = AsnIoOpen("strm.out", "w");
    fp = FileOpen("result.out", "w");

    amp=AsnLexTReadModule(aip);
    AsnTxtReadValFile(amp, aipprint, aipout);

    aip = AsnIoClose(aip);
    aipout = AsnIoClose(aipout);
    aipprint = AsnIoClose(aipprint);
    FileClose(fp);
    system("cat strm.out");
    return 0;
}


/*****************************************************************************
*
*    void AsnTxtReadValFile(amp, aip, aipout)
*      reads a file ofv values
*      prints to aipout if aipout != NULL
*
*****************************************************************************/
void AsnTxtReadValFile (AsnModulePtr amp, AsnIoPtr aip, AsnIoPtr aipout)

{
      AsnTypePtr atp;
      DataVal value;
      Boolean read_value, print_value, restart;
      Int4 baseCtr=0;
      AsnTypePtr last_atp = NULL;
      AsnTypePtr parent_atp = NULL;
      Int2 last_indent = -1;
      AsnTypePtr stack[50];
      Int2 i;

      for (i = 0; i < sizeof(stack)/sizeof(stack[0]); i++)
          stack[i] = NULL;

      if (aipout != NULL)
            print_value = TRUE;
      else
            print_value = FALSE;


      if (print_value)
            read_value = TRUE;
      else
            read_value = FALSE;


      atp = NULL;
      restart = FALSE;

      while ((atp = AsnTxtReadId(aip, amp, atp)) != NULL)
      {
```

```
            if (restart == TRUE)
            {

                  if (print_value)          /* new line */
                  {
                        GAsnPrintNewLine(aipout);
                        GAsnPrintNewLine(aipout);
                  }
                  restart = FALSE;
            }

            if (read_value)
            {
                  if (! AsnTxtReadVal(aip, atp, &value))
                        {
                        return;
                  }
                  if (print_value)
                  {
                        if (! GinoAsnTxtWrite(aipout, atp, &value, parent_atp,
&baseCtr))
                        {
                              return;
                        }
                  }

                  AsnKillValue(atp, &value);
            }
            else
            {
                  if (! AsnTxtReadVal(aip, atp, NULL))
                        return;
            }

            if (! aip->type_indent)       /* finished reading an object */
            {
                  atp = NULL;               /* restart */
                  restart = TRUE;
            }
            stack[aip->type_indent] = atp;
            parent_atp = aip->type_indent <= 0 ? NULL :
stack[aip->type_indent-1];

            last_indent = aip->type_indent;
      }
      return;
}
```

GINOPRINT.C

```
/*  ginoprint.c
*
==========================================================================
===
*
*                            PUBLIC DOMAIN NOTICE
*               National Center for Biotechnology Information
*
*  This software/database is a "United States Government Work" under the
*  terms of the United States Copyright Act.  It was written as part of
*  the author's official duties as a United States Government employee and
*  thus cannot be copyrighted.  This software/database is freely available
*  to the public for use. The National Library of Medicine and the U.S.
*  Government have not placed any restriction on its use or reproduction.
*
*  Although all reasonable efforts have been taken to ensure the accuracy
*  and reliability of the software and data, the NLM and the U.S.
*  Government do not and cannot warrant the performance or results that
*  may be obtained by using this software or data. The NLM and the U.S.
*  Government disclaim all warranties, express or implied, including
*  warranties of performance, merchantability or fitness for any particular
*  purpose.
*
*  Please cite the author in any work or product based on this material.
*
*
==========================================================================
===
*
* File Name:  asnprint.c
*
* Author:  James Ostell
*
* Version Creation Date: 3/4/91
*
* $Revision: 2.13 $
*
* File Description:
*   Routines for printing ASN.1 value notation (text) messages and
*     ASN.1 module specifications
*
* Modifications:
* --------------------------------------------------------------------------
* Date     Name        Description of modification
* -------  ----------  -----------------------------------------------------
* 3/4/91   Kans        Stricter typecasting for GNU C and C++
*
*
==========================================================================
==
*/

/*****************************************************************************
*
```

```
*   ginoprint.c
*     print routines for asn1 objects
*
*****************************************************************************/

#include "asnbuild.h"

Boolean GAsnPrintStrStore PROTO((ByteStorePtr bsp, AsnIoPtr aip));
void GAsnPrintReal PROTO((FloatHi realvalue, AsnIoPtr aip));
void GAsnPrintInteger PROTO((Int4 theInt, AsnIoPtr aip));
Boolean GAsnPrintStrStore (ByteStorePtr bsp, AsnIoPtr aip);
void GAsnPrintChar (char theChar, AsnIoPtr aip);
void GAsnPrintBoolean (Boolean value, AsnIoPtr aip);
void GAsnPrintOctets (ByteStorePtr ssp, AsnIoPtr aip);
void GAsnPrintIndent (Boolean increase, AsnIoPtr aip);
void GAsnPrintType (AsnTypePtr atp, AsnIoPtr aip);
Boolean GAsnPrintString (CharPtr the_string, AsnIoPtr aip);
void GAsnPrintOpenStruct (AsnIoPtr aip, AsnTypePtr atp);
void GAsnPrintCloseStruct (AsnIoPtr aip, AsnTypePtr atp);
void GAsnPrintNewLine (AsnIoPtr aip);

typedef struct IndexManagement {
      AsnTypePtr atp;
      Int2 idType;
      Int4 value;
      struct IndexManagement PNTR next;
} IndexManager, PNTR IndexManagerPtr;

IndexManagerPtr masterIndex = NULL;
Int4 baseCount = 0;

#define ID_ID        0
#define ID_PRIMITIVE 1
#define ID_VALUE     2

/* return an ID if one has already been created, else create a new one */
static Int4 GetUniqueID(AsnTypePtr atp, Int2 idType)
{
      IndexManagerPtr index;

      /* search for pre-existing index */
      for (index = masterIndex; index != NULL; index = index->next)
      {
            if (atp == index->atp && idType == index->idType)
            {
                  return index->value;
            }
      }

      index = (IndexManagerPtr) MemNew(sizeof(IndexManager));
      index->atp = atp;
      index->idType = idType;
      index->value = baseCount++;
      index->next = masterIndex;
      masterIndex = index;
```

```
        return index->value;
}

static void GAsnPrintNodeID(AsnTypePtr parent_atp, AsnTypePtr atp, Int2
idType, AsnIoPtr aip)
{

        Char str[50];

/*
        sprintf (str, "%ld ", (long) atp + isPrimitive);
*/
        switch (idType) {
        case ID_ID:
                  if (! (parent_atp==NULL))
                  {
                    sprintf (str, "%ld x ", GetUniqueID(parent_atp, idType));
                    GAsnPrintString(str, aip);
                  }
                  break;
        case ID_PRIMITIVE:
                  sprintf (str, "%ld x ", GetUniqueID(atp, ID_ID));
                  GAsnPrintString(str, aip);
                  break;
        case ID_VALUE:
                  sprintf (str, "%ld x ", GetUniqueID(atp, ID_PRIMITIVE));
                  GAsnPrintString(str, aip);
        }

        sprintf (str, "%ld ", GetUniqueID(atp, idType));
        GAsnPrintString(str, aip);
}

/**************************************************************************
*
*   void GinoAsnTxtWrite(aip, atp, valueptr, parent_atp, base)
*
**************************************************************************/
Boolean LIBCALL  GinoAsnTxtWrite (AsnIoPtr aip, AsnTypePtr atp, DataValPtr
dvp, AsnTypePtr
parent_atp, Int4Ptr base)
{
        Int2 isa, i;
        AsnTypePtr atp2;
        AsnValxNodePtr avnp;
        Boolean done, terminalvalue, firstvalue;
        Char str[50], temp[50];

        terminalvalue = TRUE;   /* most are terminal values */
        if ((! aip->indent_level) && (aip->typestack[0].type == NULL))
                firstvalue = TRUE;    /* first call to this routine */
        else
                firstvalue = FALSE;

        if (! AsnTypeValidateOut(aip, atp, dvp))
                return FALSE;
```

```
	atp2 = AsnFindBaseType(atp);
	isa = atp2->type->isa;
	if (ISA_STRINGTYPE(isa))
		isa = GENERALSTRING_TYPE;

	if (((isa == SEQ_TYPE) || (isa == SET_TYPE) ||
		(isa == SEQOF_TYPE) || (isa == SETOF_TYPE))
		&& (dvp->intvalue == END_STRUCT))
	{
		GAsnPrintCloseStruct(aip, atp);
		return TRUE;
	}

	if (! aip->first[aip->indent_level])
		GAsnPrintNewLine(aip);
	else
		aip->first[aip->indent_level] = FALSE;

	atp2 = atp;
	if (firstvalue)		/* first item, need ::= */
	{
		while ((atp2->name == NULL) || (IS_LOWER(*atp2->name)))
			atp2 = atp2->type;	/* find a Type Reference */
	}

	if (atp2->name != NULL)
	{
		GAsnPrintNodeID(parent_atp, atp /* atp2 */, ID_ID, aip);
		GAsnPrintString(atp2->name, aip);	/* put the element name */
		if (IS_LOWER(*atp2->name))
		{
			GAsnPrintChar('\n', aip);
		}
		else
		{
			GAsnPrintChar(' ', aip);
		}
/*

		if (IS_LOWER(*atp2->name))
			GAsnPrintChar(' ', aip);
		else
			GAsnPrintString(" ::= ", aip);
*/
	}

	if (isa == CHOICE_TYPE)	/* show nothing but name on same line */
	{
		GAsnPrintNodeID(NULL, atp, ID_PRIMITIVE, aip);
		GAsnPrintString("CHOICE", aip);
		if ((aip->type_indent))
		{
			isa = AsnFindBaseIsa(aip->typestack[aip->type_indent -
1].type);
			if ((isa != SEQOF_TYPE) && (isa != SETOF_TYPE))
```

```
            {
                  GAsnPrintIndent(TRUE, aip);
                  AsnTypeSetIndent(TRUE, aip, atp);
                  GAsnPrintNewLine(aip);
            }
            else
                  AsnTypeSetIndent(TRUE, aip, atp);
      }
      else
            AsnTypeSetIndent(TRUE, aip, atp);
      aip->first[aip->indent_level] = TRUE;
      return TRUE;
}

switch (isa)
{
      case SEQ_TYPE:
            GAsnPrintNodeID(NULL, atp, ID_PRIMITIVE, aip);
            GAsnPrintString("SEQUENCE", aip);
            GAsnPrintNewLine(aip);
            if (dvp->intvalue == START_STRUCT)    /* open brace */
                  GAsnPrintOpenStruct(aip, atp);
            else
            {
                  AsnIoErrorMsg(aip, 18 );
                  return FALSE;
            }
            terminalvalue = FALSE;
            break;
      case SET_TYPE:
            GAsnPrintNodeID(NULL, atp, ID_PRIMITIVE, aip);
            GAsnPrintString("SET", aip);
            GAsnPrintNewLine(aip);
            if (dvp->intvalue == START_STRUCT)    /* open brace */
                  GAsnPrintOpenStruct(aip, atp);
            else
            {
                  AsnIoErrorMsg(aip, 18 );
                  return FALSE;
            }
            terminalvalue = FALSE;
            break;
      case SEQOF_TYPE:
            GAsnPrintNodeID(NULL, atp, ID_PRIMITIVE, aip);
            GAsnPrintString("SEQUENCE OF", aip);
            GAsnPrintNewLine(aip);
            if (dvp->intvalue == START_STRUCT)    /* open brace */
                  GAsnPrintOpenStruct(aip, atp);
            else
            {
                  AsnIoErrorMsg(aip, 18 );
                  return FALSE;
            }
            terminalvalue = FALSE;
            break;
      case SETOF_TYPE:
```

```
        GAsnPrintNodeID(NULL, atp, ID_PRIMITIVE, aip);
        GAsnPrintString("SET OF", aip);
        GAsnPrintNewLine(aip);
        if (dvp->intvalue == START_STRUCT)   /* open brace */
              GAsnPrintOpenStruct(aip, atp);
        else
        {
              AsnIoErrorMsg(aip, 18 );
              return FALSE;
        }
        terminalvalue = FALSE;
        break;
case BOOLEAN_TYPE:
        GAsnPrintNodeID(NULL, atp, ID_PRIMITIVE, aip);
        GAsnPrintString("BOOLEAN", aip);
        GAsnPrintNewLine(aip);
        GAsnPrintNodeID(NULL, atp, ID_VALUE, aip);
        GAsnPrintBoolean(dvp->boolvalue, aip);
        break;
case INTEGER_TYPE:
        GAsnPrintNodeID(NULL, atp, ID_PRIMITIVE, aip);
        GAsnPrintString("INTEGER", aip);
        GAsnPrintNewLine(aip);
        GAsnPrintNodeID(NULL, atp, ID_VALUE, aip);
        atp2 = AsnFindBaseType(atp);  /* check for names */
        avnp = (AsnValxNodePtr) atp2->branch;
        done = FALSE;
        while (avnp != NULL)
        {
              if (dvp->intvalue == avnp->intvalue)
              {
                    GAsnPrintString(avnp->name, aip);
                    done = TRUE;
                    avnp = NULL;
              }
              else
                    avnp = avnp->next;
        }
        if (! done)     /* no name */
              GAsnPrintInteger(dvp->intvalue, aip);
        break;
case ENUM_TYPE:
        GAsnPrintNodeID(NULL, atp, ID_PRIMITIVE, aip);
        GAsnPrintString("ENUMERATED", aip);
        GAsnPrintNewLine(aip);
        GAsnPrintNodeID(NULL, atp, ID_VALUE, aip);
        atp2 = AsnFindBaseType(atp);  /* check for names */
        avnp = (AsnValxNodePtr) atp2->branch;
        done = FALSE;
        while (avnp != NULL)
        {
              if (dvp->intvalue == avnp->intvalue)
              {
                    GAsnPrintString(avnp->name, aip);
                    done = TRUE;
                    avnp = NULL;
```

```
                        }
                        else
                              avnp = avnp->next;
                  }
                  if (! done)    /* no name */
                        GAsnPrintInteger(dvp->intvalue, aip);
                  break;
            case REAL_TYPE:
                  GAsnPrintNodeID(NULL, atp, ID_PRIMITIVE, aip);
                  GAsnPrintString("REAL", aip);
                  GAsnPrintNewLine(aip);
                  GAsnPrintNodeID(NULL, atp, ID_VALUE, aip);
                  GAsnPrintReal(dvp->realvalue, aip);
                  break;
            case GENERALSTRING_TYPE:
                  GAsnPrintNodeID(NULL, atp, ID_PRIMITIVE, aip);
                  GAsnPrintString("VisibleString", aip);
                  GAsnPrintNewLine(aip);
                  GAsnPrintNodeID(NULL, atp, ID_VALUE, aip);
                  GAsnPrintChar('\"', aip);
/*                if (! GAsnPrintString((CharPtr) dvp->ptrvalue, aip))
                        return FALSE;
*/
                  sprintf(temp,((CharPtr) dvp->ptrvalue));
                  for (i=0; i<=strlen(temp); i++)
                  {
                        if (temp[i]==' ')
                              temp[i]='_';
                  }
                  GAsnPrintString(temp, aip);
                  GAsnPrintChar('\"', aip);
                  break;
            case NULL_TYPE:
                  GAsnPrintNodeID(NULL, atp, ID_PRIMITIVE, aip);
                  GAsnPrintString("NULL", aip);
                  GAsnPrintNewLine(aip);
                  GAsnPrintNodeID(NULL, atp, ID_VALUE, aip);
                  GAsnPrintString("NULL", aip);
                  break;
            case OCTETS_TYPE:
                  GAsnPrintNodeID(NULL, atp, ID_PRIMITIVE, aip);
                  GAsnPrintString("OCTET", aip);
                  GAsnPrintNewLine(aip);
                  GAsnPrintNodeID(NULL, atp, ID_VALUE, aip);
                  GAsnPrintOctets((ByteStorePtr) dvp->ptrvalue, aip);
                  break;
            case STRSTORE_TYPE:
                  GAsnPrintNodeID(NULL, atp, ID_PRIMITIVE, aip);
                  GAsnPrintString("StringStore", aip);
                  GAsnPrintNewLine(aip);
                  GAsnPrintNodeID(NULL, atp, ID_VALUE, aip);
                  if (! GAsnPrintStrStore((ByteStorePtr) dvp->ptrvalue, aip))
                        return FALSE;
                  break;
            default:
/*                AsnIoErrorMsg(aip, 19, AsnErrGetTypeName(atp->name));
```

```
*/                return FALSE;
      }

      if ((terminalvalue) && (aip->type_indent))   /* pop out of choice nests
*/
      {
            if (AsnFindBaseIsa(aip->typestack[aip->type_indent - 1].type) ==
CHOICE_TYPE)
            {
                  if (aip->type_indent >= 2)
                        isa = AsnFindBaseIsa(aip->typestack[aip->type_indent -
2].type);
                  else
                        isa = NULL_TYPE;    /* just fake it */
                  if ((isa != SETOF_TYPE) && (isa != SEQOF_TYPE))
                        GAsnPrintIndent(FALSE, aip);
                  AsnTypeSetIndent(FALSE, aip, atp);
            }
      }
      return TRUE;

}

/**************************************************************************
*
*   void GAsnPrintModule(amp, aip)
*
**************************************************************************/
void GAsnPrintModule (AsnModulePtr amp, AsnIoPtr aip)

{
      AsnTypePtr atp;
      Boolean firstone;
      CharPtr from;

      GAsnPrintString(amp->modulename, aip);
      GAsnPrintString(" DEFINITIONS ::=", aip);
      GAsnPrintNewLine(aip);
      GAsnPrintString("BEGIN", aip);
      GAsnPrintNewLine(aip);
      GAsnPrintNewLine(aip);

      atp = amp->types;           /* check for EXPORTS */
      firstone = TRUE;
      while (atp != NULL)
      {
            if (atp->exported == TRUE)
            {
                  if (firstone)
                        GAsnPrintString("EXPORTS ", aip);
                  else
                  {
                        GAsnPrintString(" ,", aip);
                        GAsnPrintNewLine(aip);
                        GAsnPrintString("        ", aip);
                  }
```

```
                GAsnPrintString(atp->name, aip);
                firstone = FALSE;
            }
            atp = atp->next;
        }
        if (! firstone)            /* got at least one */
        {
            GAsnPrintString(" ;", aip);
            GAsnPrintNewLine(aip);
            GAsnPrintNewLine(aip);
        }

        atp = amp->types;           /* check for IMPORTS */
        firstone = TRUE;
        from = NULL;
        while (atp != NULL)
        {
            if (atp->imported == TRUE)
            {
                if (firstone)
                    GAsnPrintString("IMPORTS ", aip);
                else
                {
                    if (StringCmp((CharPtr) atp->branch, from))    /* new
FROM */
                    {
                        GAsnPrintString(" FROM ", aip);
                        GAsnPrintString(from, aip);
                    }
                    else
                        GAsnPrintString(" ,", aip);
                    GAsnPrintNewLine(aip);
                    GAsnPrintString("        ", aip);
                }
                GAsnPrintString(atp->name, aip);
                firstone = FALSE;
                from = (CharPtr) atp->branch;
            }
            atp = atp->next;
        }
        if (! firstone)            /* got at least one */
        {
            GAsnPrintString(" FROM ", aip);
            GAsnPrintString(from, aip);
            GAsnPrintString(" ;", aip);
            GAsnPrintNewLine(aip);
            GAsnPrintNewLine(aip);
        }

        atp = amp->types;
        while (atp != NULL)
        {
            if (! atp->imported)
            {
                GAsnPrintString(atp->name, aip);
                GAsnPrintString(" ::= ", aip);
```

```
                GAsnPrintType(atp, aip);
                GAsnPrintNewLine(aip);
                GAsnPrintNewLine(aip);
            }
            atp = atp->next;
        }
        GAsnPrintString("END", aip);
        GAsnPrintNewLine(aip);
        return;
}

/**************************************************************************
*
*   void GAsnPrintType(atp, aip)
*     prints a type starting at current line position
*     (assumes name already printed)
*
**************************************************************************/
void GAsnPrintType (AsnTypePtr atp, AsnIoPtr aip)

{
        AsnValxNodePtr avnp;
        AsnTypePtr atp2;
        Boolean first;

        if (atp->tagclass != TAG_NONE)      /* print tag, if any */
        {
            GAsnPrintChar('[', aip);
            GAsnPrintChar(' ', aip);
            switch (atp->tagclass)
            {
                case TAG_UNIVERSAL:
                    GAsnPrintString("UNIVERSAL ", aip);
                    break;
                case TAG_APPLICATION:
                    GAsnPrintString("APPLICATION ", aip);
                    break;
                case TAG_PRIVATE:
                    GAsnPrintString("PRIVATE ", aip);
                    break;
                default:        /* context dependent, do nothing */
                    break;
            }
            GAsnPrintInteger((Int4)atp->tagnumber, aip);
            GAsnPrintChar(' ', aip);
            GAsnPrintChar(']', aip);
            GAsnPrintChar(' ', aip);

            if (atp->implicit)
                GAsnPrintString("IMPLICIT ", aip);
        }

        GAsnPrintString(atp->type->name, aip);    /* print the type name */

        if (atp->branch != NULL)        /* sub types ? */
        {
```

```
switch (atp->type->isa)
{
      case SETOF_TYPE:
      case SEQOF_TYPE:
            GAsnPrintChar(' ', aip);
            GAsnPrintType((AsnTypePtr) atp->branch, aip);
            break;
      case INTEGER_TYPE:
      case ENUM_TYPE:
            GAsnPrintChar(' ', aip);
            GAsnPrintOpenStruct(aip, atp);
            avnp = (AsnValxNodePtr)atp->branch;
            first = TRUE;
            aip->first[aip->indent_level] = FALSE;
            while (avnp != NULL)
            {
                  if (! first)
                        GAsnPrintNewLine(aip);
                  else
                        first = FALSE;
                  GAsnPrintString(avnp->name, aip);
                  GAsnPrintChar(' ', aip);
                  GAsnPrintChar('(', aip);
                  GAsnPrintInteger(avnp->intvalue, aip);
                  GAsnPrintChar(')', aip);
                  avnp = avnp->next;
            }
            GAsnPrintCloseStruct(aip, atp);
            break;
      case SEQ_TYPE:
      case SET_TYPE:
      case CHOICE_TYPE:
            GAsnPrintChar(' ', aip);
            GAsnPrintOpenStruct(aip, atp);
            atp2 = (AsnTypePtr) atp->branch;
            first = TRUE;
            aip->first[aip->indent_level] = FALSE;
            while (atp2 != NULL)
            {
                  if (! first)
                        GAsnPrintNewLine(aip);
                  else
                        first = FALSE;

                  if (atp2->name != NULL)
                  {
                        GAsnPrintString(atp2->name, aip);
                        GAsnPrintChar(' ', aip);
                  }
                  GAsnPrintType(atp2, aip);
                  atp2 = atp2->next;
            }
            GAsnPrintCloseStruct(aip, atp);
            break;
      default:                  /* everything else */
            break;            /* do nothing */
```

```
        }
    }

    if (atp->optional)
        GAsnPrintString(" OPTIONAL", aip);

    if (atp->hasdefault)
    {
        GAsnPrintString(" DEFAULT ", aip);
        avnp = atp->defaultvalue;
        while (! (VALUE_ISA_DEFAULT(avnp->valueisa)))
            avnp = avnp->next;
        switch (avnp->valueisa)
        {
            case VALUE_ISA_PTR:
                GAsnPrintChar('\"', aip);
                GAsnPrintString(avnp->name, aip);
                GAsnPrintChar('\"', aip);
                break;
            case VALUE_ISA_BOOL:
                GAsnPrintBoolean((Boolean)avnp->intvalue, aip);
                break;
            case VALUE_ISA_INT:
                GAsnPrintInteger(avnp->intvalue, aip);
                break;
            case VALUE_ISA_REAL:
                GAsnPrintReal(avnp->realvalue, aip);
                break;
            default:
                GAsnPrintString("Error", aip);
                break;
        }
    }
}

/**************************************************************************
*
*   Boolean GAsnPrintStrStore(bsp, aip)
*
**************************************************************************/
Boolean GAsnPrintStrStore (ByteStorePtr bsp, AsnIoPtr aip)

{
    Char buf[101];
    Uint4 len, tlen;

    if (aip->type & ASNIO_CARRIER)          /* pure iterator */
        return TRUE;

    BSSeek(bsp, 0, SEEK_SET);      /* seek to start */
    len = BSLen(bsp);
    GAsnPrintChar('\"', aip);
    while (len)
    {
        if (len < 100)
            tlen = len;
```

```
            else
                  tlen = 100;
            BSRead(bsp, buf, tlen);
            buf[tlen] = '\0';
            if (! GAsnPrintString(buf, aip))
                  return FALSE;
            len -= tlen;
      }
      GAsnPrintChar('\"', aip);
      return TRUE;
}
/*****************************************************************************
*
*   void GAsnPrintReal(realvalue, aip)
*
*****************************************************************************/
void GAsnPrintReal (FloatHi realvalue, AsnIoPtr aip)

{
      FloatHi thelog, mantissa;
      int characteristic;
      int   ic;
      long  im;
      char tbuf[30];
      Boolean minus;

      if (aip->type & ASNIO_CARRIER)          /* pure iterator */
            return;

      if (realvalue == 0.0)
      {
            ic = 0;
            im = 0;
      }
      else
      {
            if (realvalue < 0.0)
            {
                  minus = TRUE;
                  realvalue = -realvalue;
            }
            else
                  minus = FALSE;

            thelog = log10((double)realvalue);
            if (thelog >= 0.0)
                  characteristic = 8 - (int)thelog;/* give it 9 significant
digits */
            else
                  characteristic = 8 + (int)ceil(-thelog);

            mantissa = realvalue * Nlm_Powi((double)10., characteristic);
            ic = -characteristic; /* reverse direction */
            im = (long) mantissa;

            /* strip trailing 0 */
```

```
            while ((im % 10L) == 0L)
            {
                  im /= 10L;
                  ic++;
            }

            if (minus)
                  im = -im;
      }
      sprintf(tbuf, "{ %ld, 10, %d }", im, ic);
      GAsnPrintString(tbuf, aip);
      return;
}

/**************************************************************************
*
*    void GAsnPrintInteger(theInt, aip)
*
**************************************************************************/
void GAsnPrintInteger (Int4 theInt, AsnIoPtr aip)

{
      char tbuf[10];

      if (aip->type & ASNIO_CARRIER)          /* pure iterator */
            return;

      sprintf(tbuf, "%ld", (long)theInt);
      GAsnPrintString(tbuf, aip);
      return;
}

/**************************************************************************
*
*    void GAsnPrintChar(theChar, aip)
*      print a single character
*
**************************************************************************/
void GAsnPrintChar (char theChar, AsnIoPtr aip)

{
      if (aip->type & ASNIO_CARRIER)          /* pure iterator */
            return;

      *(aip->linebuf + aip->linepos) = theChar;
      aip->linepos++;
      aip->offset++;
      return;
}

/**************************************************************************
*
*    void GAsnPrintBoolean(value, aip)
*
**************************************************************************/
void GAsnPrintBoolean (Boolean value, AsnIoPtr aip)
```

```
{
      if (aip->type & ASNIO_CARRIER)           /* pure iterator */
            return;

      if (value)
            GAsnPrintString("TRUE", aip);
      else
            GAsnPrintString("FALSE", aip);
      return;
}

/**************************************************************************
*
*   void GAsnPrintOctets(ssp, aip)
*
**************************************************************************/
void GAsnPrintOctets (ByteStorePtr ssp, AsnIoPtr aip)

{
      Int2 value, tval, ctr;
      Char buf[101];

      if (aip->type & ASNIO_CARRIER)           /* pure iterator */
            return;

      GAsnPrintChar('\'', aip);

      BSSeek(ssp, 0, SEEK_SET);   /* go to start of bytestore */
      ctr = 0;
      buf[100] = '\0';

                              /* break it up into lines if necessary */
      while ((value = BSGetByte(ssp)) != -1)
      {
            tval = value / 16;
            if (tval < 10)
                  buf[ctr] = (Char)(tval + '0');
            else
                  buf[ctr] = (Char)(tval - 10 + 'A');
            ctr++;
            tval = value - (tval * 16);
            if (tval < 10)
                  buf[ctr] = (Char)(tval + '0');
            else
                  buf[ctr] = (Char)(tval - 10 + 'A');
            ctr++;
            if (ctr == 100)
            {
                GAsnPrintString(buf, aip);
                  ctr = 0;
            }
      }
      if (ctr)
      {
            buf[ctr] = '\0';
```

```
            GAsnPrintString(buf, aip);
      }

      GAsnPrintChar('\'', aip);
      GAsnPrintChar('H', aip);
      return;
}

/**************************************************************************
*
*   void GAsnPrintIndent(increase, aip)
*      increase or decrease indent level
*
**************************************************************************/
void GAsnPrintIndent (Boolean increase, AsnIoPtr aip)

{
      Int1 offset,
            curr_indent;
      BoolPtr tmp;
      int decr, isa;


      if (increase)
      {
            aip->indent_level++;
            curr_indent = aip->indent_level;
            if (curr_indent == aip->max_indent)   /* expand indent levels */
            {
                  tmp = aip->first;
                  aip->first = (BoolPtr) MemNew((sizeof(Boolean) *
(aip->max_indent +
10)));
                  MemCopy(aip->first, tmp, (size_t)(sizeof(Boolean) *
aip->max_indent));
                  MemFree(tmp);
                  aip->max_indent += 10;
            }
            aip->first[curr_indent] = TRUE;     /* set to first time */
            offset = curr_indent * aip->tabsize;

            if (! (aip->type & ASNIO_CARRIER))
            {
                  while (aip->linepos < offset)
                  {
                        *(aip->linebuf + aip->linepos) = ' ';
                        aip->linepos++;
                  }
                  aip->offset = aip->linepos + (aip->linebuf -
(CharPtr)aip->buf);
            }
      }
      else
      {
            offset = aip->indent_level * aip->tabsize;
            curr_indent = aip->type_indent;
```

```
            decr = 1;   /* always backup indent for named element */
            do
            {
                  if (aip->indent_level)
                        aip->indent_level -= decr;
                  if (curr_indent)
                        curr_indent--;
                  isa = NULL_TYPE;        /* fake key */
                  if ((aip->indent_level) && (curr_indent))
                  {
                        isa = AsnFindBaseIsa(aip->typestack[curr_indent -
1].type);
                        if (aip->typestack[curr_indent-1].type->name != NULL)
                              decr = 1;     /* indent for named choices as
elements */
                        else
                              decr = 0;     /* not referenced choice objects
*/
                  }
            } while (isa == CHOICE_TYPE);

            if (aip->linepos == offset)    /* nothing written yet */
            {
                  curr_indent = aip->indent_level * aip->tabsize;
                  while (offset >= curr_indent)
                  {
                        offset--;
                        if (! (aip->type & ASNIO_CARRIER))
                        {
                              if ((offset >= 0) && (aip->linebuf[offset] != '
'))
                                    curr_indent = 127;
                        }
                  }
                  offset++;
                  aip->linepos = offset;
                  aip->offset = aip->linepos + (aip->linebuf -
(CharPtr)aip->buf);
            }
            if (! aip->indent_level)   /* level 0 - no commas */
                  aip->first[0] = TRUE;
      }
      return;
}

/**************************************************************************
*
*   void GAsnPrintNewLine(aip)
*       end a line in the print buffer
*       indent to the proper level on the next line
*
**************************************************************************/
void GAsnPrintNewLine (AsnIoPtr aip)

{
      Int1 tpos, indent;
```

```
	CharPtr tmp;
	Boolean do_print = TRUE;

	if (aip->linepos == 0)		/* nothing in buffer yet */
		return;

	if (! (aip->type & ASNIO_CARRIER))		/* really printing */
	{
		tpos = aip->indent_level * aip->tabsize;
		if (tpos == aip->linepos)	/* just an empty indent? */
		{
			do_print = FALSE;	/* assume that's the case */
			for (tmp = aip->linebuf; tpos != 0; tpos--, tmp++)
			{
				if (*tmp != ' ')
				{
					do_print = TRUE;  /* set sentinel */
					break;
				}
			}
		}

		if (do_print)	/* not an empty indent */
		{
			tmp = aip->linebuf + aip->linepos;
			if (aip->first[aip->indent_level] == FALSE)	/* not first
line of struct */
			{
								/* add
commas */
#ifdef JAE_NOWAY
				*tmp = ' '; tmp++;
				*tmp = ','; tmp++;
#endif /* JAE_NOWAY */
			}
			else if (aip->linepos)		/* is first line, remove
trailing blanks */
			{
								/* if just
indented */
				tmp--;
				while ((*tmp == ' ') && (tmp > aip->linebuf))
					tmp--;
				tmp++;
			}
			*tmp = '\0';
			aip->linepos = tmp - aip->linebuf;
			aip->offset = tmp - (CharPtr)aip->buf;
			AsnIoPuts(aip);
		}
	}

	if ((do_print) && (aip->indent_level))	/* level 0 never has commas */
		aip->first[aip->indent_level] = FALSE;

	if (! (aip->type & ASNIO_CARRIER))	/* really printing */
	{
```

```
            tmp = aip->linebuf;
            indent = aip->indent_level * aip->tabsize;
            indent = 0;
            for (tpos = 0; tpos < indent; tpos++, tmp++)
                  *tmp = ' ';
            aip->linepos = tpos;
            aip->offset += tpos;
      }
      return;
}
/*****************************************************************************
*
*   Boolean GAsnPrintString(str, aip)
*
*****************************************************************************/
Boolean GAsnPrintString (CharPtr the_string, AsnIoPtr aip)

{
      Uint4 stringlen;
      register int templen;
      Int1 first = 1;
      register CharPtr current, str;
      Boolean indent_state;
      int bad_char, bad_char_ctr = 0;
fprintf (stderr, "%s", the_string);

      if (aip->type & ASNIO_CARRIER)          /* pure iterator */
            return TRUE;

      str = the_string;
      stringlen = StringLen(str);
      indent_state = aip->first[aip->indent_level];


                              /* break it up into lines if necessary */
      while (stringlen)
      {
            if (! first)                /* into multiple lines */
            {
                  aip->first[aip->indent_level] = TRUE;   /* no commas */
                  GAsnPrintNewLine(aip);
                  aip->offset -= aip->linepos;
                  aip->linepos = 0;
            }
            first = 0;

            templen = (int)(aip->linelength - aip->linepos);

            if (stringlen <= (Uint4)templen)     /* it fits in remaining space
*/
                  templen = (int) stringlen;
            else
                  templen = GAsnPrintGetWordBreak(str, templen);

            current = aip->linebuf + aip->linepos;
            stringlen -= (Uint4)templen;
```

```
            aip->linepos += templen;
            aip->offset += templen;
            while (templen)
            {
                  if ((aip->fix_non_print < 2) && ((*str < ' ') || (*str >
'~')))
                  {
                        if (! bad_char_ctr)
                              bad_char = (int)(*str);
                        bad_char_ctr++;

                        *str = '#';   /* replace with # */
                  }
                  *current = *str;
                  if (*str == '\"')     /* must double quotes */
                  {
                        current++; aip->linepos++; aip->offset++;
                        *current = '\"';
                  }
                  current++; str++; templen--;
            }
      }
      aip->first[aip->indent_level] = indent_state;   /* reset indent state */
      if ((bad_char_ctr) && (aip->fix_non_print == 0))
      {
            AsnIoErrorMsg(aip, 106, bad_char, the_string);
      }
      return TRUE;
}

/**************************************************************************
*
*   void GAsnPrintCharBlock(str, aip)
*      prints string on line if there is room
*      if not prints on next line with no indent.
*
**************************************************************************/
void GAsnPrintCharBlock (CharPtr str, AsnIoPtr aip)

{
      Uint4 stringlen;
      Boolean indent_state;
      Int1 templen;
      CharPtr current;

      if (aip->type & ASNIO_CARRIER)           /* pure iterator */
            return;

      stringlen = StringLen(str);
      templen = (Int1)(aip->linelength - aip->linepos);
      indent_state = aip->first[aip->indent_level];

      if (stringlen > (Uint4)templen)     /* won't fit on line */
      {
            aip->first[aip->indent_level] = TRUE;   /* no commas */
            GAsnPrintNewLine(aip);
```

```
            aip->linepos = 0;       /* no indent on broken string */
        }

        current = aip->linebuf + aip->linepos;
        MemCopy(current, str, (size_t)stringlen);
        aip->linepos += (Int2) stringlen;
        aip->offset += (Int2) stringlen;
        aip->first[aip->indent_level] = indent_state;   /* reset indent state */
        return;
}

/*****************************************************************************
*
*   int GAsnPrintGetWordBreak(str, maxlen)
*       return length (<= maxlen) of str to next white space
*
*****************************************************************************/
int GAsnPrintGetWordBreak (CharPtr str, int maxlen)

{
        CharPtr tmp;
        int len;
        Uint4 stringlen;

        stringlen = StringLen(str);
        if (stringlen <= (Uint4)maxlen)
            return (int) stringlen;

        tmp = str + maxlen;     /* point just PAST the end of region */
        len = maxlen + 1;
        while ((len) && (! IS_WHITESP(*tmp)))
        {
            len--; tmp--;
        }
        while ((len) && (IS_WHITESP(*tmp)))
        {
            len--;              /* move past white space */
            tmp--;
        }
        if (len < 1)          /* never found any whitespace or only 1 space */
            len = maxlen;     /* have to break a word */

        return len;
}

/*****************************************************************************
*
*   GAsnPrintOpenStruct(aip, atp)
*
*****************************************************************************/
void GAsnPrintOpenStruct (AsnIoPtr aip, AsnTypePtr atp)

{
#ifdef NOWAY_JAE
      GAsnPrintChar('{', aip);
#endif /* NOWAY_JAE */
```

```
	GAsnPrintIndent(TRUE, aip);
	AsnTypeSetIndent(TRUE, aip, atp);
	GAsnPrintNewLine(aip);
	aip->first[aip->indent_level] = TRUE;
	return;
}

/**************************************************************************
*
*   GAsnPrintCloseStruct(aip, atp)
*
**************************************************************************/
void GAsnPrintCloseStruct (AsnIoPtr aip, AsnTypePtr atp)

{
/*
	GAsnPrintChar(' ', aip);
	GAsnPrintChar('}', aip);
*/
	GAsnPrintIndent(FALSE, aip);
	AsnTypeSetIndent(FALSE, aip, atp);
	return;
}
```

PROJ.H

```
/*******************************
*
*  Header for Proj
*
*  by: Gino Celia, Jr
*
*******************************/


void buildTree(Widget w, char* infilename);


void ButtonCallback (Widget w, XtPointer clientData, XtPointer callData);
void Button1Callback (Widget w, XtPointer clientData, XtPointer callData);
void D1OKCallback (Widget w, XtPointer clientData, XtPointer callData);
void Button2Callback (Widget w, XtPointer clientData, XtPointer callData);
void D2OKCallback (Widget w, XtPointer clientData, XtPointer callData);
void Button3Callback (Widget w, XtPointer clientData, XtPointer callData);
void D3OKCallback (Widget w, XtPointer clientData, XtPointer callData);
void Button11Callback (Widget w, XtPointer clientData, XtPointer callData);
void D11OKCallback (Widget w, XtPointer clientData, XtPointer callData);
void Button13Callback (Widget w, XtPointer clientData, XtPointer callData);
void Button14Callback (Widget w, XtPointer clientData, XtPointer callData);
void Button15Callback (Widget w, XtPointer clientData, XtPointer callData);
void Button16Callback (Widget w, XtPointer clientData, XtPointer callData);

void ShowSelectedWidget (Widget w, XtPointer clientData, XEvent *event,
Boolean *flag);
void ValueChangedCallback (Widget w, XtPointer clientData, XtPointer
callData);
void makeButtons(Widget w);

void createTreeCascadeL (Widget w);
void createTreeCascadeS (Widget w);

void ExitCallback (Widget w, XtPointer clientData, XtPointer callData);
void LoadCallback (Widget w, XtPointer clientData, XtPointer callData);
void SaveCallback (Widget w, XtPointer clientData, XtPointer callData);
void OKCallback (Widget w, XtPointer clientData, XtPointer callData);
void CancelCallback (Widget w, XtPointer clientData, XtPointer callData);

Widget createMenu (Widget w);

void createFilePane (Widget w);
void createHelpPane (Widget w);

void copyFiles (char* infilename, char* outfilename);
```

PROJ.C

```
/*****************************************************************************
 *
 *   proj.c
 *
 *   Final thesis project by : LT Gino Celia, Jr., US Navy
 *
 *   Graphical ASN.1 database integration and conflict resolution tool
 *
 *   September, 1994
 *
 *****************************************************************************/



/*****************************************************************************
 *         Portions of this code are from the book:
 *
 *           The X Window System: Programming and Applications with Xt
 *           Second OSF/Motif Edition
 *         by
 *           Douglas Young
 *           Prentice Hall, 1994
 *
 *         Copyright 1994 by Prentice Hall
 *         All Rights Reserved
 *
 *****************************************************************************/

#include <ncbi.h>
#include <ncbiwin.h>

/* this stuff covered in ncbi.h and ncbiwin.h */
/*
#include <stdio.h>
#include <Xm/Xm.h>
#include <Xm/MainW.h>
#include <Xm/Form.h>
#include <Xm/RowColumn.h>
#include <Xm/Label.h>
#include <Xm/CascadeB.h>
#include <Xm/PushB.h>
#include <Xm/ScrolledW.h>
*/

#include <Xm/MessageB.h>
#include <Xm/DrawnB.h>
#include <Xm/FileSB.h>
#include <Xm/ArrowB.h>
#include "proj.h"
#include "Tree.h"

typedef struct {
      Widget old, new;
```

```
   } Dialog1Widgets;

typedef struct {
      Widget oldLabel, newType;
   } Dialog2Widgets;

typedef struct {
      Widget old1, old2, new;
   } Dialog3Widgets;

typedef struct {
      Widget old, mult, new;
   } Dialog11Widgets;


Widget       tree1,
       tree2,
       form1,
       form2;
int    activeTree = 1,
       t1RecNum   = 0,
       t2RecNum   = 0,
       t1LastRec   = 0,
       t2LastRec   = 0;
char   *t1_infile = "tree1.dat";
char   *t2_infile = "tree2.dat";
char   *t1_tempfile = "tree1temp.dat";
char   *t2_tempfile = "tree2temp.dat";
char   *t1_outfile = "tree1out.dat";
char   *t2_outfile = "tree2out.dat";


void main ( int argc, char **argv )
{
    Arg             wargs[15];
    int             n=0;
    Widget        shell,
              mainwindow,
              menu,
              shell1,
              shell2,
              form;
    XtAppContext  app;

   /* initialize Xt */

   shell = XtVaAppInitialize (&app, "Tree", NULL, 0,
                              &argc, argv, NULL,
                         XmNgeometry, "900x340+200+25",
                         NULL);

   /* create two popup shells */
   shell1= XtVaCreatePopupShell ("shell1", topLevelShellWidgetClass,
                            shell,
                          NULL);
   shell2= XtVaCreatePopupShell ("shell2", topLevelShellWidgetClass,
```

```
                              shell,
                            NULL);

   /* Create a manager window for the main shell */
   mainwindow = XtVaCreateManagedWidget ("mainwindow",
xmMainWindowWidgetClass,
                                    shell, NULL);
   /* Create the menu bar */
   menu = createMenu (mainwindow);

   /* Create a form for each popup and the mainwindow*/
   n = 0;
   form = XtCreateManagedWidget ("form", xmFormWidgetClass, mainwindow, wargs,
n);
   form1 = XtCreateManagedWidget ("form1", xmFormWidgetClass, shell1, wargs,
n);
   form2 = XtCreateManagedWidget ("form2", xmFormWidgetClass, shell2, wargs,
n);

   /* Specify the widgets for the mainwindow */
   XtVaSetValues (mainwindow,
               XmNmenuBar, menu,
               XmNworkWindow, form,
               NULL);

   copyFiles (t1_infile, t1_tempfile);
   copyFiles (t2_infile, t2_tempfile);

   tree1 = XsCreateScrolledTree (form1, "tree1", NULL, 0);
   tree2 = XsCreateScrolledTree (form2, "tree2", NULL, 0);

   buildTree (tree1, t1_infile);
   buildTree (tree2, t2_infile);

   XtManageChild (tree1);
   XtManageChild (tree2);

   /* Create the buttons */
   makeButtons (form);

   /* Popup the popups */
   XtPopup (shell1, XtGrabNone);
   XtPopup (shell2, XtGrabNone);

   /* Realize everything */
   XtRealizeWidget (shell);
   XtAppMainLoop (app);
}


/*  buildTree creates the nodes for the tree widget */

void buildTree(Widget w, char* infilename)
{
   char        parent[500],
              parentLabel[500],
```

```
            child[500],
           childLabel[500];
int              recNum = 0,
           success = 0,
           lastrec = 0;
FILE*        infile;

infile = fopen (infilename, "r");
fseek (infile, 0, 0);

/* get the right record number */
   if (strcmp(XtName(w), "tree1") == 0)
    recNum = t1RecNum;
   else
    recNum = t2RecNum;


  while (fscanf (infile, "%s %s %s %s", &parent, &parentLabel, &child,
&childLabel) != EOF)
   {
     /* check to see if we're on the right record */
     if (strcmp(parent, "0") == 0)
     {
      lastrec = atoi(childLabel);
      if (atoi(childLabel) == recNum)
      {
         success = 1;
      }
      else
         success = 0;
     }

     /* build the tree */
     if (success)
     {
         Widget p,
              c;

        /*
         * If a parent identifier was read, check to see if this name
         * has already been used as a widget. If so, use the existing
         * widget as the supernode of the given child.
         */

         if (parent)
           p = XtNameToWidget (w, parent);

         if (!p)
         {
            /*
           * Otherwise, create a new widget for this node.
           */

           p = XtVaCreateManagedWidget (parent,
                                xmDrawnButtonWidgetClass,
                                w,
```

```
                                XtVaTypedArg, XmNlabelString,
                                XmRString,    parentLabel,
                                strlen(parentLabel)+1,
                                NULL);
          }

         /*
          * If a child identifier was read, check to see if this name
          * has already been used as a widget. If so, use the existing
          * widget as the subnode of the given parent.
          */

          if (child)
            c = XtNameToWidget (w, child);

          if (!c)
          {

             /*
            * Otherwise, create a new widget for this node.
            */

            c = XtVaCreateManagedWidget (child,
                                xmDrawnButtonWidgetClass,
                                w,
                                XmNsuperNode, p,
                                XtVaTypedArg, XmNlabelString,
                                XmRString,    childLabel,
                                strlen(childLabel)+1,
                                NULL );
          }
       }
    }


    fclose (infile);

    if (strcmp(XtName(w), "tree1") == 0)
        t1LastRec = lastrec;
       else
        t2LastRec = lastrec;
 }

/* the button callbacks */
/* default callback */
void ButtonCallback (Widget w, XtPointer clientData, XtPointer callData)
{
    static Widget  dialog = NULL;
    char             temp[30] = " ";

    dialog = XmCreateInformationDialog (w, "dialog", NULL, 0);
    sprintf (temp, "%s selected", XtName (w));
    XtVaSetValues (dialog,
              XmNmessageString, XmStringCreateSimple(temp),
              XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL,
              NULL);
```

```
   XtManageChild (dialog);
}

/* change name */
void Button1Callback (Widget w, XtPointer clientData, XtPointer callData)
{
   Widget     dialog1 = NULL;
   Widget     rc;
   Dialog1Widgets  *widgets;

   widgets = (Dialog1Widgets *) XtMalloc (sizeof (Dialog1Widgets));
   dialog1 = XmCreateMessageDialog (w, "dialog1", NULL, 0);
   XtUnmanageChild (XmMessageBoxGetChild (dialog1, XmDIALOG_SYMBOL_LABEL));
   XtUnmanageChild (XmMessageBoxGetChild (dialog1, XmDIALOG_MESSAGE_LABEL));
   rc = XtVaCreateManagedWidget ("rc", xmRowColumnWidgetClass, dialog1,
                           XmNnumColumns, 2,
                           XmNpacking, XmPACK_COLUMN,
                           XmNorientation, XmVERTICAL,
                           NULL);
   XtCreateManagedWidget ("Old Name", xmLabelWidgetClass, rc, NULL, 0);
   XtCreateManagedWidget ("New Name", xmLabelWidgetClass, rc, NULL, 0);
   widgets->old = XtCreateManagedWidget ("old", xmTextFieldWidgetClass, rc,
NULL, 0);
   widgets->new = XtCreateManagedWidget ("new", xmTextFieldWidgetClass, rc,
NULL, 0);
   XtVaSetValues (dialog1,
              XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL,
              NULL);
   XtAddCallback (dialog1, XmNokCallback, D1OKCallback, (XtPointer) widgets);
   XtManageChild (dialog1);
}

void D1OKCallback (Widget w, XtPointer clientData, XtPointer callData)
{
   Dialog1Widgets  *widgets;
   char        parent[500],
             parentLabel[500],
               child[500],
             childLabel[500],
             tempold [500],
             tempnew [500];
   FILE*       tempfile;
   FILE*       outfile;


   widgets = (Dialog1Widgets *) clientData;
   strcpy (tempold, XmTextFieldGetString(widgets->old));
   strcpy (tempnew, XmTextFieldGetString(widgets->new));
   if (activeTree == 1)
   {
      tempfile = fopen (t1_tempfile, "r");
      outfile = fopen (t1_outfile, "w");
      while (fscanf (tempfile, "%s %s %s %s", &parent, &parentLabel, &child,
&childLabel) != EOF)
      {
       if (strcmp (parentLabel, tempold) == 0)
```

```
            strcpy (parentLabel, tempnew);
         if (strcmp (childLabel, tempold) == 0)
            strcpy (childLabel, tempnew);
         fprintf (outfile, "%s %s %s %s \n", parent, parentLabel, child,
childLabel);
      }
      fclose (outfile);
      fclose (tempfile);
      copyFiles (t1_outfile, t1_tempfile);
      tree1 = XsCreateScrolledTree (form1, "tree1", NULL, 0);
      buildTree(tree1, t1_tempfile);
      XtManageChild(tree1);
   }
   else
   {
      tempfile = fopen (t2_tempfile, "r");
      outfile = fopen (t2_outfile, "w");
      while (fscanf (tempfile, "%s %s %s %s", &parent, &parentLabel, &child,
&childLabel) != EOF)
      {
       if (strcmp (parentLabel, tempold) == 0)
          strcpy (parentLabel, tempnew);
       if (strcmp (childLabel, tempold) == 0)
          strcpy (childLabel, tempnew);
       fprintf (outfile, "%s %s %s %s \n", parent, parentLabel, child,
childLabel);
      }
      fclose (outfile);
      fclose (tempfile);
      copyFiles (t2_outfile, t2_tempfile);
      tree2 = XsCreateScrolledTree (form2, "tree2", NULL, 0);
      buildTree(tree2, t2_tempfile);
      XtManageChild(tree2);
   }
}

/* change type */
void Button2Callback (Widget w, XtPointer clientData, XtPointer callData)
{
   Widget     dialog2 = NULL;
   Widget     rc;
   Dialog2Widgets  *widgets;

   widgets = (Dialog2Widgets *) XtMalloc (sizeof (Dialog2Widgets));
   dialog2 = XmCreateMessageDialog (w, "dialog2", NULL, 0);
   XtUnmanageChild (XmMessageBoxGetChild (dialog2, XmDIALOG_SYMBOL_LABEL));
   XtUnmanageChild (XmMessageBoxGetChild (dialog2, XmDIALOG_MESSAGE_LABEL));
   rc = XtVaCreateManagedWidget ("rc", xmRowColumnWidgetClass, dialog2,
                        XmNnumColumns, 2,
                        XmNpacking, XmPACK_COLUMN,
                        XmNorientation, XmVERTICAL,
                        NULL);
   XtCreateManagedWidget ("Node Name", xmLabelWidgetClass, rc, NULL, 0);
   XtCreateManagedWidget ("New Type", xmLabelWidgetClass, rc, NULL, 0);
   widgets->oldLabel = XtCreateManagedWidget ("oldLabel",
xmTextFieldWidgetClass, rc, NULL,
```

```
0);
   widgets->newType = XtCreateManagedWidget ("newType",
xmTextFieldWidgetClass, rc,
NULL, 0);
   XtVaSetValues (dialog2,
              XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL,
              NULL);
   XtAddCallback (dialog2, XmNokCallback, D2OKCallback, (XtPointer) widgets);
   XtManageChild (dialog2);
}

void D2OKCallback (Widget w, XtPointer clientData, XtPointer callData)
{
   Dialog2Widgets  *widgets;
   char        parent1[500],
             parentLabel1[500],
               child1[500],
             childLabel1[500],
             parent2[500],
             parentLabel2[500],
               child2[500],
             childLabel2[500],
             tempold [500],
             tempnew [500],
             templabel [500];
   FILE*       tempfile;
   FILE*       outfile;


   widgets = (Dialog2Widgets *) clientData;
   strcpy (tempold, XmTextFieldGetString(widgets->oldLabel));
   strcpy (tempnew, XmTextFieldGetString(widgets->newType));
   if (activeTree == 1)
   {
      tempfile = fopen (t1_tempfile, "r");
      outfile = fopen (t1_outfile, "w");
      while (fscanf (tempfile, "%s %s %s %s", &parent1, &parentLabel1,
&child1, &childLabel1) !=
EOF)
      {
       if (strcmp (childLabel1, tempold) == 0)
       {
          strcpy (templabel, child1);
          fseek (tempfile, 0, 0);
          while (fscanf (tempfile, "%s %s %s %s", &parent2, &parentLabel2,
&child2,
&childLabel2) != EOF)
          {
             if (strcmp (parent2, templabel) == 0)
             {
                strcpy (childLabel2, tempnew);
             }
             fprintf (outfile, "%s %s %s %s \n", parent2, parentLabel2,
child2, childLabel2);
          }
       }
```

```
      }
      fclose (outfile);
      fclose (tempfile);
      copyFiles (t1_outfile, t1_tempfile);
      tree1 = XsCreateScrolledTree (form1, "tree1", NULL, 0);
      buildTree(tree1, t1_tempfile);
      XtManageChild(tree1);
   }
   else
   {
      tempfile = fopen (t2_tempfile, "r");
      outfile = fopen (t2_outfile, "w");
      while (fscanf (tempfile, "%s %s %s %s", &parent1, &parentLabel1,
&child1, &childLabel1) !=
EOF)
      {
       if (strcmp (childLabel1, tempold) == 0)
       {
          strcpy (templabel, child1);
          fseek (tempfile, 0, 0);
          while (fscanf (tempfile, "%s %s %s %s", &parent2, &parentLabel2,
&child2,
&childLabel2) != EOF)
          {
             if (strcmp (parent2, templabel) == 0)
             {
                strcpy (childLabel2, tempnew);
             }
             fprintf (outfile, "%s %s %s %s \n", parent2, parentLabel2,
child2, childLabel2);
          }
       }
      }
      fclose (outfile);
      fclose (tempfile);
      copyFiles (t2_outfile, t2_tempfile);
      tree2 = XsCreateScrolledTree (form2, "tree2", NULL, 0);
      buildTree(tree2, t2_tempfile);
      XtManageChild(tree2);
   }
}

/* horiz concat */
void Button3Callback (Widget w, XtPointer clientData, XtPointer callData)
{
   Widget     dialog3 = NULL;
   Widget     rc;
   Dialog3Widgets  *widgets;

   widgets = (Dialog3Widgets *) XtMalloc (sizeof (Dialog3Widgets));
   dialog3 = XmCreateMessageDialog (w, "dialog3", NULL, 0);
   XtUnmanageChild (XmMessageBoxGetChild (dialog3, XmDIALOG_SYMBOL_LABEL));
   XtUnmanageChild (XmMessageBoxGetChild (dialog3, XmDIALOG_MESSAGE_LABEL));
   rc = XtVaCreateManagedWidget ("rc", xmRowColumnWidgetClass, dialog3,
                         XmNnumColumns, 2,
                         XmNpacking, XmPACK_COLUMN,
```

```
                        XmNorientation, XmVERTICAL,
                        NULL);
   XtCreateManagedWidget ("Old Label 1", xmLabelWidgetClass, rc, NULL, 0);
   XtCreateManagedWidget ("Old Label 2", xmLabelWidgetClass, rc, NULL, 0);
   XtCreateManagedWidget ("New Label", xmLabelWidgetClass, rc, NULL, 0);
   widgets->old1 = XtCreateManagedWidget ("old1", xmTextFieldWidgetClass, rc,
NULL, 0);
   widgets->old2 = XtCreateManagedWidget ("old2", xmTextFieldWidgetClass, rc,
NULL, 0);
   widgets->new  = XtCreateManagedWidget ("new", xmTextFieldWidgetClass, rc,
NULL, 0);
   XtVaSetValues (dialog3,
            XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL,
            NULL);
   XtAddCallback (dialog3, XmNokCallback, D3OKCallback, (XtPointer) widgets);
   XtManageChild (dialog3);
}

void D3OKCallback (Widget w, XtPointer clientData, XtPointer callData)
{
   Dialog3Widgets  *widgets;
   int             success = 0;
   char        parent1[500],
            parentLabel1[500],
              child1[500],
            childLabel1[500],
            parent2[500],
            parentLabel2[500],
              child2[500],
            childLabel2[500],
            tempold1 [500],
            tempold2 [500],
            tempnew [500];
   FILE*       tempfile;
   FILE*       outfile;


   widgets = (Dialog3Widgets *) clientData;
   printf ("%s %s %s \n", XmTextFieldGetString(widgets->old1),
                     XmTextFieldGetString(widgets->old2),
                     XmTextFieldGetString(widgets->new));
   strcpy (tempold1, XmTextFieldGetString(widgets->old1));
   strcpy (tempold2, XmTextFieldGetString(widgets->old2));
   strcpy (tempnew, XmTextFieldGetString(widgets->new));

   if (activeTree == 1)
   {
      tempfile = fopen (t1_tempfile, "r");
      outfile = fopen (t1_outfile, "w");
      success = 0;
      while (fscanf (tempfile, "%s %s %s %s", &parent1, &parentLabel1,
&child1, &childLabel1) !=
EOF)
      {
       if (strcmp (childLabel1, tempold1) == 0)
       {
```

```
          fseek (tempfile, 0, 0);
          while (fscanf (tempfile, "%s %s %s %s", &parent2, &parentLabel2,
&child2,
&childLabel2) != EOF)
          {
             if (strcmp (childLabel2, tempold2) == 0)
             {
                success = 1;
             }
          }
       }
      }
      if (success)
      {
       printf ("Both items found!!! \n");
      }
      else
       printf ("One or both item(s) not found \n");
      fclose (outfile);
      fclose (tempfile);
/*      copyFiles (t1_outfile, t1_tempfile);
      tree1 = XsCreateScrolledTree (form1, "tree1", NULL, 0);
      buildTree(tree1, t1_tempfile);
      XtManageChild(tree1);
*/   }
/*   else
   {
      tempfile = fopen (t2_tempfile, "r");
      outfile = fopen (t2_outfile, "w");
      while (fscanf (tempfile, "%s %s %s %s", &parent1, &parentLabel1,
&child1, &childLabel1) !=
EOF)
      {
       if (strcmp (childLabel1, tempold) == 0)
       {
          strcpy (templabel, child1);
          fseek (tempfile, 0, 0);
          while (fscanf (tempfile, "%s %s %s %s", &parent2, &parentLabel2,
&child2,
&childLabel2) != EOF)
          {
             if (strcmp (parent2, templabel) == 0)
             {
                strcpy (childLabel2, tempnew);
             }
             fprintf (outfile, "%s %s %s %s \n", parent2, parentLabel2,
child2, childLabel2);
          }
       }
      }
      fclose (outfile);
      fclose (tempfile);
      copyFiles (t2_outfile, t2_tempfile);
      tree2 = XsCreateScrolledTree (form2, "tree2", NULL, 0);
      buildTree(tree2, t2_tempfile);
      XtManageChild(tree2);
```

```
   }
*/}

/* change units */
void Button11Callback (Widget w, XtPointer clientData, XtPointer callData)
{
   Widget     dialog11 = NULL;
   Widget     rc;
   Dialog11Widgets  *widgets;

   widgets = (Dialog11Widgets *) XtMalloc (sizeof (Dialog11Widgets));
   dialog11 = XmCreateMessageDialog (w, "dialog11", NULL, 0);
   XtUnmanageChild (XmMessageBoxGetChild (dialog11, XmDIALOG_SYMBOL_LABEL));
   XtUnmanageChild (XmMessageBoxGetChild (dialog11, XmDIALOG_MESSAGE_LABEL));
   rc = XtVaCreateManagedWidget ("rc", xmRowColumnWidgetClass, dialog11,
                          XmNnumColumns, 2,
                          XmNpacking, XmPACK_COLUMN,
                          XmNorientation, XmVERTICAL,
                          NULL);
   XtCreateManagedWidget ("Old Label", xmLabelWidgetClass, rc, NULL, 0);
   XtCreateManagedWidget ("Multiplier", xmLabelWidgetClass, rc, NULL, 0);
   XtCreateManagedWidget ("New Label", xmLabelWidgetClass, rc, NULL, 0);
   widgets->old = XtCreateManagedWidget ("old", xmTextFieldWidgetClass, rc,
NULL, 0);
   widgets->mult = XtCreateManagedWidget ("mult", xmTextFieldWidgetClass, rc,
NULL, 0);
   widgets->new  = XtCreateManagedWidget ("new", xmTextFieldWidgetClass, rc,
NULL, 0);
   XtVaSetValues (dialog11,
              XmNdialogStyle, XmDIALOG_FULL_APPLICATION_MODAL,
              NULL);
   XtAddCallback (dialog11, XmNokCallback, D11OKCallback, (XtPointer)
widgets);
   XtManageChild (dialog11);
}

void D11OKCallback (Widget w, XtPointer clientData, XtPointer callData)
{
   Dialog11Widgets  *widgets;
   char        parent1[500],
             parentLabel1[500],
               child1[500],
             childLabel1[500],
             parent2[500],
             parentLabel2[500],
               child2[500],
             childLabel2[500],
             templabel1[500],
             templabel2[500],
             tempold [500],
             tempnew [500];
   float       tempmult;
   FILE*       tempfile;
   FILE*       outfile;
```

```
   widgets = (Dialog11Widgets *) clientData;
   printf ("%s %s %s %.2f \n", XmTextFieldGetString(widgets->old),
                    XmTextFieldGetString(widgets->mult),
                    XmTextFieldGetString(widgets->new),
                    atof(XmTextFieldGetString(widgets->mult)));
   strcpy (tempold, XmTextFieldGetString(widgets->old));
   tempmult = atof (XmTextFieldGetString(widgets->mult));
   strcpy (tempnew, XmTextFieldGetString(widgets->new));
  /* if (activeTree == 1)
   {
      tempfile = fopen (t1_tempfile, "r");
      outfile = fopen (t1_outfile, "w");
      strcpy (templabel1," ");
      strcpy (templabel2," ");
      while (fscanf (tempfile, "%s %s %s %s", &parent1, &parentLabel1,
&child1, &childLabel1) !=
EOF)
      {
       if (strcmp (childLabel1, tempold) == 0)
       {
          strcpy (templabel1, child1);
          strcpy (childLabel1, tempnew);
       }
       else
          if (strcmp (parent1, templabel1) == 0)
             strcpy (templabel2, child1);
          else
             if (strcmp (parent1, templabel2) == 0)
              strcpy (childLabel1, ecvt ((atof (childLabel1) * tempmult), 2,
NULL, NULL));
       fprintf (outfile, "%s %s %s %s \n", parent1, parentLabel1, child1,
childLabel1);
      }
      fclose (outfile);
      fclose (tempfile);
      copyFiles (t1_outfile, t1_tempfile);
      tree1 = XsCreateScrolledTree (form1, "tree1", NULL, 0);
      buildTree(tree1, t1_tempfile);
      XtManageChild(tree1);
   }
   else
   {
      tempfile = fopen (t2_tempfile, "r");
      outfile = fopen (t2_outfile, "w");
      while (fscanf (tempfile, "%s %s %s %s", &parent1, &parentLabel1,
&child1, &childLabel1) !=
EOF)
      {
       if (strcmp (childLabel1, tempold) == 0)
       {
          strcpy (templabel, child1);
          fseek (tempfile, 0, 0);
          while (fscanf (tempfile, "%s %s %s %s", &parent2, &parentLabel2,
&child2,
&childLabel2) != EOF)
          {
```

```
            if (strcmp (parent2, templabel) == 0)
            {
               strcpy (childLabel2, tempnew);
            }
            fprintf (outfile, "%s %s %s %s \n", parent2, parentLabel2,
child2, childLabel2);
          }
        }
      }
      fclose (outfile);
      fclose (tempfile);
      copyFiles (t2_outfile, t2_tempfile);
      tree2 = XsCreateScrolledTree (form2, "tree2", NULL, 0);
      buildTree(tree2, t2_tempfile);
      XtManageChild(tree2);
   } */
}


/* go to first record */
void Button13Callback (Widget w, XtPointer clientData, XtPointer callData)
{
   if (activeTree == 1)
   {
      if (t1RecNum != 0)
      {
       t1RecNum = 0;
       tree1 = XsCreateScrolledTree (form1, "tree1", NULL, 0);
       buildTree(tree1, t1_tempfile);
       XtManageChild(tree1);
      }
   }
   else
   {
      if (t2RecNum != 0)
      {
       t2RecNum = 0;
       tree2 = XsCreateScrolledTree (form2, "tree2", NULL, 0);
       buildTree(tree2, t2_tempfile);
       XtManageChild(tree2);
      }
   }
}

/* go to previous record */
void Button14Callback (Widget w, XtPointer clientData, XtPointer callData)
{
   if (activeTree == 1)
   {
      t1RecNum--;
      if (t1RecNum < 0)
       t1RecNum = 0;
      else
      {
       tree1 = XsCreateScrolledTree (form1, "tree1", NULL, 0);
       buildTree(tree1, t1_tempfile);
```

```
        XtManageChild(tree1);
      }
   }
   else
   {
      t2RecNum--;
      if (t2RecNum < 0)
       t2RecNum = 0;
      else
      {
       tree2 = XsCreateScrolledTree (form2, "tree2", NULL, 0);
       buildTree(tree2, t2_tempfile);
       XtManageChild(tree2);
      }
   }
}

/* go to next record */
void Button15Callback (Widget w, XtPointer clientData, XtPointer callData)
{
   if (activeTree == 1)
   {
      t1RecNum++;
      if (t1RecNum > t1LastRec)
       t1RecNum = t1LastRec;
      else
      {
        tree1 = XsCreateScrolledTree (form1, "tree1", NULL, 0);
        buildTree(tree1, t1_tempfile);
        XtManageChild(tree1);
      }
   }
   else
   {
      t2RecNum++;
      if (t2RecNum > t2LastRec)
       t2RecNum = t2LastRec;
      else
      {
        tree2 = XsCreateScrolledTree (form2, "tree2", NULL, 0);
        buildTree(tree2, t2_tempfile);
        XtManageChild(tree2);
      }
   }
}

/* go to last record */
void Button16Callback (Widget w, XtPointer clientData, XtPointer callData)
{
   if (activeTree == 1)
   {
      if (t1RecNum != t1LastRec)
      {
       t1RecNum = t1LastRec;
       tree1 = XsCreateScrolledTree (form1, "tree1", NULL, 0);
       buildTree(tree1, t1_tempfile);
```

```
        XtManageChild(tree1);
       }
    }
    else
    {
       if (t2RecNum != t2LastRec)
       {
        t2RecNum = t2LastRec;
        tree2 = XsCreateScrolledTree (form2, "tree2", NULL, 0);
        buildTree(tree2, t2_tempfile);
        XtManageChild(tree2);
       }
    }
}

void ShowSelectedWidget (Widget w, XtPointer clientData,
                   XEvent *event, Boolean *flag)
{
   printf ("button pressed\n");
   printf ("%s selected \n", XtName(w));
}

void ValueChangedCallback (Widget w, XtPointer clientData, XtPointer callData)
{
   XmToggleButtonCallbackStruct *cbs =
              ( XmToggleButtonCallbackStruct * ) callData;

   if ((strcmp ("toggle1", XtName(w)) == 0) && (cbs->set))
      activeTree = 1;
   else
      if ((strcmp ("toggle2", XtName(w)) == 0) && (cbs->set))
       activeTree = 2;
}

/* makeButtons creates the control buttons */
void makeButtons (Widget w)
{

   Widget   label1,
          label2,
          rowcol1,
          rowcol2,
          sep,
          button1,
          button2,
          button3,
          button4,
          button5,
          button6,
          button7,
          button8,
          button9,
          button10,
          button11,
          button12,
          radio,
```

```
        label3,
        toggle1,
        toggle2,
        recordBox,
        label4,
        button13,
        button14,
        button15,
        button16;

label1 = XtVaCreateManagedWidget ("label1", xmLabelWidgetClass,
                            w,
                            XmNtopAttachment, XmATTACH_WIDGET,
                              XmNtopWidget, w,
                              XmNleftAttachment, XmATTACH_WIDGET,
                              XmNleftWidget, w,
                            NULL);

label2 = XtVaCreateManagedWidget ("label2", xmLabelWidgetClass,
                            w,
                            XmNtopAttachment, XmATTACH_WIDGET,
                            XmNtopWidget, w,
                            XmNrightAttachment, XmATTACH_WIDGET,
                            XmNrightWidget, w,
                            NULL);

rowcol1 = XtVaCreateManagedWidget ("rowcol1", xmRowColumnWidgetClass,
                            w,
                              XmNtopAttachment, XmATTACH_WIDGET,
                              XmNtopWidget, label1,
                            XmNbottomAttachment, XmATTACH_POSITION,
                              XmNbottomPosition, 84,
                              XmNleftAttachment, XmATTACH_WIDGET,
                              XmNleftWidget, w,
                            XmNrightAttachment, XmATTACH_NONE,
                            XmNorientation, XmVERTICAL,
                            XmNisAligned, TRUE,
                            XmNentryAlignment, XmALIGNMENT_CENTER,
                            XmNnumColumns, 2,
                            XmNpacking, XmPACK_COLUMN,
                                  NULL);

rowcol2 = XtVaCreateManagedWidget ("rowcol2", xmRowColumnWidgetClass,
                            w,
                            XmNtopAttachment, XmATTACH_OPPOSITE_WIDGET,
                              XmNtopWidget, rowcol1,
                            XmNbottomAttachment, XmATTACH_OPPOSITE_WIDGET,
                            XmNbottomWidget, rowcol1,
                              XmNrightAttachment, XmATTACH_WIDGET,
                              XmNrightWidget, w,
                            XmNorientation, XmVERTICAL,
                            XmNisAligned, TRUE,
                            XmNentryAlignment, XmALIGNMENT_CENTER,
                            XmNnumColumns, 2,
                            XmNpacking, XmPACK_COLUMN,
                            NULL);
```

```
    sep = XtVaCreateManagedWidget ("sep", xmSeparatorWidgetClass,
                                   w,
                                   XmNleftAttachment, XmATTACH_WIDGET,
                                     XmNleftWidget, rowcol1,
                                   XmNrightAttachment, XmATTACH_WIDGET,
                                   XmNrightWidget, rowcol2,
                                   XmNtopAttachment, XmATTACH_OPPOSITE_WIDGET,
                                   XmNtopWidget, rowcol1,
XmNbottomAttachment, XmATTACH_OPPOSITE_WIDGET,
                                   XmNbottomWidget, rowcol1,
                                   NULL);

    button1 = XtVaCreateManagedWidget ("button1", xmPushButtonWidgetClass,
                                       rowcol1,
                                       NULL);

    XtAddCallback (button1, XmNactivateCallback, Button1Callback, NULL);

    button2 = XtVaCreateManagedWidget ("button2", xmPushButtonWidgetClass,
                                       rowcol1,
                                       NULL);

    XtAddCallback (button2, XmNactivateCallback, Button2Callback, NULL);

    button3 = XtVaCreateManagedWidget ("button3", xmPushButtonWidgetClass,
                                       rowcol1,
                                       NULL);

    XtAddCallback (button3, XmNactivateCallback, Button3Callback, NULL);

    button4 = XtVaCreateManagedWidget ("button4", xmPushButtonWidgetClass,
                                       rowcol1,
                                       NULL);

    XtAddCallback (button4, XmNactivateCallback, ButtonCallback, NULL);

    button5 = XtVaCreateManagedWidget ("button5", xmPushButtonWidgetClass,
                                       rowcol1,
                                       NULL);

    XtAddCallback (button5, XmNactivateCallback, ButtonCallback, NULL);

    button6 = XtVaCreateManagedWidget ("button6", xmPushButtonWidgetClass,
                                       rowcol1,
                                       NULL);

    XtAddCallback (button6, XmNactivateCallback, ButtonCallback, NULL);

    button7 = XtVaCreateManagedWidget ("button7", xmPushButtonWidgetClass,
                                       rowcol1,
                                       NULL);

    XtAddCallback (button7, XmNactivateCallback, ButtonCallback, NULL);

    button8 = XtVaCreateManagedWidget ("button8", xmPushButtonWidgetClass,
```

```
                                rowcol1,
                                NULL);

    XtAddCallback (button8, XmNactivateCallback, ButtonCallback, NULL);

    button9 = XtVaCreateManagedWidget ("button9", xmPushButtonWidgetClass,
                                rowcol1,
                                NULL);

    XtAddCallback (button9, XmNactivateCallback, ButtonCallback, NULL);

    button10 = XtVaCreateManagedWidget ("button10", xmPushButtonWidgetClass,
                                rowcol1,
                                NULL);

    XtAddCallback (button10, XmNactivateCallback, ButtonCallback, NULL);

    button11 = XtVaCreateManagedWidget ("button11", xmPushButtonWidgetClass,
                                rowcol1,
                                NULL);

    XtAddCallback (button11, XmNactivateCallback, Button11Callback, NULL);

    button12 = XtVaCreateManagedWidget ("button12", xmPushButtonWidgetClass,
                                rowcol1,
                                NULL);

    XtAddCallback (button12, XmNactivateCallback, ButtonCallback, NULL);


    radio = XtVaCreateManagedWidget ("rowcol2", xmRowColumnWidgetClass,
                               w,
                               XmNtopAttachment, XmATTACH_POSITION,
                                 XmNtopPosition, 85,
                               XmNbottomAttachment, XmATTACH_WIDGET,
                               XmNbottomWidget, w,
                                 XmNrightAttachment, XmATTACH_NONE,
                                 XmNleftAttachment, XmATTACH_WIDGET,
                                 XmNleftWidget, w,
                               XmNradioBehavior, TRUE,
                               XmNorientation, XmHORIZONTAL,
                               NULL);

    label3 = XtVaCreateManagedWidget ("label3", xmLabelWidgetClass,
                                radio,
                                XmNhighlightOnEnter, FALSE,
                                NULL);

    toggle1 = XtVaCreateManagedWidget ("toggle1", xmToggleButtonWidgetClass,
                                radio,
                                XmNset, TRUE,
                                NULL);

    XtAddCallback (toggle1, XmNvalueChangedCallback, ValueChangedCallback,
NULL);
```

```
   toggle2 = XtVaCreateManagedWidget ("toggle2", xmToggleButtonWidgetClass,
                                radio,
                                NULL);

   XtAddCallback (toggle2, XmNvalueChangedCallback, ValueChangedCallback,
NULL);

   recordBox = XtVaCreateManagedWidget ("rowcol3", xmRowColumnWidgetClass,
                               w,
                               XmNtopAttachment, XmATTACH_OPPOSITE_WIDGET,
                                 XmNtopWidget, radio,
                               XmNbottomAttachment, XmATTACH_OPPOSITE_WIDGET,
                               XmNbottomWidget, radio,
                                 XmNrightAttachment, XmATTACH_WIDGET,
                               XmNrightWidget, w,
                                 XmNleftAttachment, XmATTACH_WIDGET,
                                 XmNleftWidget, radio,
                               XmNorientation, XmHORIZONTAL,

                               NULL);

   label4 = XtVaCreateManagedWidget ("label4", xmLabelWidgetClass,
                                recordBox,
                                XmNhighlightOnEnter, FALSE,
                                NULL);

   button13 = XtVaCreateManagedWidget ("button13", xmArrowButtonWidgetClass,
                                 recordBox,
                                 XmNarrowDirection, XmARROW_LEFT,
                                 NULL);

   XtAddCallback (button13, XmNactivateCallback, Button13Callback, NULL);

   button14 = XtVaCreateManagedWidget ("button14", xmArrowButtonWidgetClass,
                                 recordBox,
                                 XmNarrowDirection, XmARROW_LEFT,
                                 NULL);

   XtAddCallback (button14, XmNactivateCallback, Button14Callback, NULL);

   button15 = XtVaCreateManagedWidget ("button15", xmArrowButtonWidgetClass,
                                 recordBox,
                                 XmNarrowDirection, XmARROW_RIGHT,
                                 NULL);

   XtAddCallback (button15, XmNactivateCallback, Button15Callback, NULL);

   button16 = XtVaCreateManagedWidget ("button16", xmArrowButtonWidgetClass,
                                 recordBox,
                                 XmNarrowDirection, XmARROW_RIGHT,
                                  NULL);

   XtAddCallback (button16, XmNactivateCallback, Button16Callback, NULL);
```

```
}


/* the menu callbacks */
/* exit */
void ExitCallback (Widget w, XtPointer clientData, XtPointer callData)
{
   copyFiles (t1_tempfile, t1_outfile);
   copyFiles (t2_tempfile, t2_outfile);
   exit(0);   /* outta here */
}

/* load */
void LoadCallback (Widget w, XtPointer clientData, XtPointer callData)
{
/*
   static Widget fileDialog = NULL;
   Arg wargs[10];
   int n;

      n=0;
      XtSetArg (wargs[n], XmNpattern, "*.dat"); n++;
      fileDialog = XmCreateFileSelectionDialog (w, "openFileDialog", NULL, 0);

      XtAddCallback (fileDialog, XmNokCallback, OKCallback, NULL);
      XtAddCallback (fileDialog, XmNcancelCallback, CancelCallback, NULL);

   XtManageChild (fileDialog);
*/
}

void OKCallback (Widget w, XtPointer clientData, XtPointer callData)
{

/*
   XmFileSelectionBoxCallbackStruct *cbs =
          (XmFileSelectionBoxCallbackStruct *) callData;

   XtUnmanageChild (w);

   XmStringGetLtoR (cbs->value, XmFONTLIST_DEFAULT_TAG, &fileName);

   printf ("%s is the parent of %s\n", XtName(XtParent(XtParent(w))),
XtName(w));

   if (strcmp(XtName (XtParent(XtParent(w))), "Tree 1") == 0)
   {
      printf ("Loading tree 1 !!!! with file %s \n", fileName);
      if (XtIsRealized (tree1))
       XtUnmanageChild (tree1);
      t1_infile = fopen (fileName, "r");
      buildTree (tree1, t1_infile);
      XtManageChild (tree1);
   }
   else
   {
```

```
        printf ("Loading tree 2 !!!! with file %s \n", fileName);
        if (XtIsRealized (tree2))
         XtUnmanageChild (tree2);
        t2_infile = fopen (fileName, "r");
        buildTree (tree2, t2_infile);
        XtManageChild (tree2);
    }
*/
}

void CancelCallback (Widget w, XtPointer clientData, XtPointer callData)
{
   XtUnmanageChild (w);
}

/* save */
void SaveCallback (Widget w, XtPointer clientData, XtPointer callData)
{
   /*save tree here*/
   if (strcmp(XtName (w), "Tree 1") == 0)
      printf ("Saving tree 1 !!!!\n");
   else
      printf ("Saving tree 2 !!!!\n");
}

/* create the menu bar */
Widget createMenu (Widget w)
{

   Widget menu;

   menu = XmCreateMenuBar (w, "menu", NULL, 0);

   createFilePane (menu);
   createHelpPane (menu);

   XtManageChild (menu);

   return (menu);

}

/* create the file pane */
void createFilePane (Widget w)
{
   Widget   cascade,
          cascade1,
          cascade2,
          submenu,
          submenu1,
          submenu2,
          button1,
          button2,
          button3;

   submenu = XmCreatePulldownMenu (w, "fileSubmenu", NULL, 0);
```

```
    cascade = XtVaCreateManagedWidget ("File", xmCascadeButtonWidgetClass,
                                w,
                                XmNsubMenuId, submenu,
                                NULL);

    submenu1 = XmCreatePulldownMenu (submenu, "loadSubmenu", NULL, 0);
    cascade1 = XtVaCreateManagedWidget ("Load", xmCascadeButtonWidgetClass,
                                submenu,
                                XmNsubMenuId, submenu1,
                                NULL);

    createTreeCascadeL (submenu1);

    submenu2 = XmCreatePulldownMenu (submenu, "saveSubmenu", NULL, 0);
    cascade2 = XtVaCreateManagedWidget ("Save", xmCascadeButtonWidgetClass,
                                submenu,
                                XmNsubMenuId, submenu2,
                                NULL);

    createTreeCascadeS (submenu2);

    button3 = XtCreateManagedWidget ("Exit", xmPushButtonWidgetClass,
                              submenu, NULL, 0);

    XtAddCallback (button3, XmNactivateCallback, ExitCallback, NULL);
}

/* create tree cascade for load menu*/
void createTreeCascadeL (Widget w)
{
    Widget button1,
         button2;


    button1 = XtCreateManagedWidget ("Tree 1", xmPushButtonWidgetClass,
                              w, NULL, 0);

    button2 = XtCreateManagedWidget ("Tree 2", xmPushButtonWidgetClass,
                              w, NULL, 0);
/*
    XtAddCallback (button1, XmNactivateCallback, LoadCallback, NULL);
    XtAddCallback (button2, XmNactivateCallback, LoadCallback, NULL);
*/
}

/* create tree cascade for save menu*/
void createTreeCascadeS (Widget w)
{
    Widget button1,
         button2;


    button1 = XtCreateManagedWidget ("Tree 1", xmPushButtonWidgetClass,
                              w, NULL, 0);
```

```
   button2 = XtCreateManagedWidget ("Tree 2", xmPushButtonWidgetClass,
                              w, NULL, 0);

   XtAddCallback (button1, XmNactivateCallback, SaveCallback, NULL);
   XtAddCallback (button2, XmNactivateCallback, SaveCallback, NULL);

}

/* create the help pane */
void createHelpPane (Widget w)
{
   Widget   cascade,
          submenu,
          button1,
          button2;

   submenu = XmCreatePulldownMenu (w, "helpSubmenu", NULL, 0);
   cascade = XtVaCreateManagedWidget ("Help", xmCascadeButtonWidgetClass,
                                 w,
                                 XmNsubMenuId, submenu,
                                 NULL);

   XtVaSetValues (w, XmNmenuHelpWidget, cascade, NULL);
}

/* write the output files */
void copyFiles (char* infilename, char* outfilename)
{
   char        parent[500],
             parentLabel[500],
               child[500],
             childLabel[500];
   FILE*       infile;
   FILE*       outfile;

   infile = fopen (infilename, "r");
   outfile = fopen (outfilename, "w");
   while (fscanf (infile, "%s %s %s %s", &parent, &parentLabel, &child,
&childLabel) != EOF)
   {
      fprintf (outfile, "%s %s %s %s \n", parent, parentLabel, child,
childLabel);
   }
   fclose (infile);
   fclose (outfile);
}
```

TREE

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!! AppDefaults file for Proj
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!! Tree widget colors
!*Tree.Background: midnight blue
!*Tree.Foreground: white

*XmForm*Background: dark slate grey
*XmForm*XmScrollBar*background: grey
*XmForm*XmDrawnButton*background: grey
*XmForm*XmDrawnButton*foreground: black

*XmMainWindow*Background: light blue
*XmMainWindow*Foreground: black

!! Tree fonts
*tree1*fontList: -*-helvetica-medium-r-normal-*-10-*-*-*-*-*-iso8859-1
*tree2*fontList: -*-helvetica-medium-r-normal-*-10-*-*-*-*-*-iso8859-1

!! Tree attachments
*form1*bottomAttachment: attach_form
*form1*topAttachment: attach_form
*form1*leftAttachment: attach_form
*form1*rightAttachment: attach_form
*form2*bottomAttachment: attach_form
*form2*topAttachment: attach_form
*form2*leftAttachment: attach_form
*form2*rightAttachment: attach_form

!! Popup shell geometry
*shell1.geometry: 600x600+25+500
*shell2.geometry: 600x600+660+500

!! Dialog default position
*XmMessageBox.defaultPosition: FALSE
*XmMessageBox.x: 500
*XmMessageBox.y: 500


!! Label strings
*label1.labelString: Resolution Commands
*label2.labelString: DML Commands
*label3.labelString: Active Tree:
*label4.labelString: Record:

!! Separator
*sep.separatorType: DOUBLE_LINE
*sep.orientation: VERTICAL

!! Button label strings
*rowcol1.button1.labelString: Change Node NAME
*rowcol1.button2.labelString: Change Node TYPE
*rowcol1.button3.labelString: Horiz CONCAT
```

```
*rowcol1.button4.labelString: Horiz SUBSET
*rowcol1.button5.labelString: Vert COLLAPSE
*rowcol1.button6.labelString: Vert EXPAND
*rowcol1.button7.labelString: Change SEQUENCE
*rowcol1.button8.labelString: OPTIONALITY
*rowcol1.button9.labelString: CHOICE
*rowcol1.button10.labelString: Change PRECISION
*rowcol1.button11.labelString: Change UNITS
*rowcol1.button12.labelString: Change EXPRESSION

!! Toggle label strings
*toggle1.labelString: Tree 1
*toggle2.labelString: Tree 2
```

SPEC1.ASN

```
--****************************************
--
--  CDB-1 data definitions
--  Gino Celia, 1994
--
--****************************************

Component-one-module DEFINITIONS ::=
BEGIN

Book-set ::= SEQUENCE OF Book       -- collection of books

Book ::= SEQUENCE {
   b-num INTEGER,              -- local key
   title VisibleString,
   author-name VisibleString,       -- last, first
   subj VisibleString OPTIONAL,
   type CHOICE {
      book SEQUENCE {
      binding ENUMERATED {
               hardcover(1),
               paperback(2) },
      num-pgs INTEGER },
      music SEQUENCE {
      medium ENUMERATED {
           record(1),
           cd(2),
           tape(3) },
        length INTEGER },           -- in minutes
      movie SEQUENCE {
      format ENUMERATED {
           beta(1),
           vhs(2),
           reel(3) },
      length INTEGER }},            -- in minutes
   language VisibleString DEFAULT "English",
   lc-num SEQUENCE {
      c-letter VisibleString,       -- one or more CAP LTRS
      f-digit VisibleString,        -- one or more digits
      s-digit VisibleString OPTIONAL,     -- one or more digits
      cuttering VisibleString },    -- author cutter number
   publisher-name VisibleString,
   publisher-addr VisibleString,    -- num, str, city, state
   checked-out BOOLEAN,             -- TRUE if in library
   cost INTEGER }                   -- orig cost in whole dollars

END
```

PRINT1.ENT

```
Holding ::= {
b-num 10,
title "Joint Military Operations: A Short History",
author-name "Beaumont, Roger A.",
subj "Military Science",
type book {
binding hardcover,
num-pgs 245
},
language "English",
lc-num {
c-letter "U",
f-digit "260",
cuttering "B43"
},
publisher-name "Greenwood Press",
publisher-addr "Westport, Connecticut",
checked-out TRUE,
cost 60
}
```

PRINT1.OUT

```
Book ::= {
  b-num 10 ,
  title "Joint Military Operations: A Short History" ,
  author-name "Beaumont, Roger A." ,
  subj "Military Science" ,
  type
    book {
      binding hardcover ,
      num-pgs 245 } ,
  language "English" ,
  lc-num {
    c-letter "U" ,
    f-digit "260" ,
    cuttering "B43" } ,
  publisher-name "Greenwood Press" ,
  publisher-addr "Westport, Connecticut" ,
  checked-out TRUE ,
  cost 60 }
```

TREE1.DAT

```
0 0 0 0
1 Holding 2a SEQUENCE
2a x 2 b-num
2a x 3 title
2a x 4 author-name
2a x 5 subj
2a x 6 type
2a x 7 language
2a x 8 lc-num
2a x 9 publisher-name
2a x 10 publisher-addr
2a x 11 checked-out
2a x 12 cost
2 x 13 INTEGER
13 x 14 10
3 x 15 VisibleString
15 x 16 "Joint_Military_Operations:A_Short_History"
4 x 17 VisibleString
17 x 18 "Beaumont,_Roger_A."
5 x 19 VisibleString
19 x 20 "Military_Science"
6 x 21 CHOICE
21 x 22 book
22 x 23 SEQUENCE
23 x 24 binding
23 x 25 num-pgs
24 x 26 ENUMERATED
26 x 27 hardcover
25 x 28 INTEGER
28 x 29 245
7 x 30 VisibleString
30 x 31 "English"
8 x 32 SEQUENCE
32 x 33 c-letter
32 x 34 f-digit
32 x 35 s-digit
32 x 36 cuttering
33 x 37 VisibleString
37 x 38 "U"
34 x 39 VisibleString
39 x 40 "260"
35 x 41 VisibleString
36 x 42 VisibleString
42 x 43 "B43"
9 x 44 VisibleString
44 x 45 "Greenwood_Press"
10 x 46 VisibleString
46 x 47 "Westport,_Connecticut"
11 x 48 BOOLEAN
48 x 49 TRUE
12 x 50 INTEGER
50 x 51 60
0 1 0 1
52 Holding 53 SEQUENCE
```

```
53 x 54 b-num
53 x 55 title
53 x 56 author-name
53 x 57 subj
53 x 58 type
53 x 59 language
53 x 60 lc-num
53 x 61 publisher-name
53 x 62 publisher-addr
53 x 63 checked-out
53 x 64 cost
54 x 65 INTEGER
65 x 66 11
55 x 67 VisibleString
67 x 68 "X_Window_System"
56 x 69 VisibleString
69 x 70 "Scheifler,_Robert_W."
57 x 71 VisibleString
71 x 72 "Computers"
58 x 73 CHOICE
73 x 74 book
74 x 75 SEQUENCE
75 x 76 binding
75 x 77 num-pgs
76 x 78 ENUMERATED
78 x 79 hardcover
77 x 78 INTEGER
78 x 79 701
59 x 80 VisibleString
80 x 81 "English"
60 x 82 SEQUENCE
82 x 83 c-letter
82 x 84 f-digit
82 x 85 s-digit
82 x 86 cuttering
83 x 87 VisibleString
87 x 88 "QA"
84 x 89 VisibleString
89 x 90 "76"
85 x 91 VisibleString
91 x 91a ".76"
86 x 92 VisibleString
92 x 93 "W56"
61 x 94 VisibleString
94 x 95 "Digital_Press"
62 x 96 VisibleString
96 x 97 "Massachusetts"
63 x 98 BOOLEAN
98 x 99 FALSE
64 x 100 INTEGER
100 x 101 65
```

LIST OF REFERENCES

Ahmed, R., et al., "Pegasus Heterogeneous Multidatabase System," *IEEE Computer*, v. 24, No. 12, pp. 19-27, 1991.

Batini, C., and Lenzerini, M., "A Methodology for Data Schema Integration in the ER Model," *IEEE Transactions on Software Engineering*, pp. 650-664, November 1984.

Batini, C., and Lenzerini, M., "A Comparative Analysis of Methodologies for Database Schema Integration," *ACM Computing Surveys*, v. 18, No. 4, pp. 323-364, 1986.

Elmasri, R., and Wiederhold, G., "Data Model Integration Using the Structural Model," paper presented at the Proc. Int. Conference on Management of Data, pp. 191-202, 1979.

Gonnet, G. H., and Tompa, F. W., "Mind Your Grammar: A New Approach to Modeling Text," paper presented at the Proceedings of the 13th VLDB Conference, pp. 339-346, 1987.

Kamel, M., "Identifying, Classifying, and Resolving Semantic Conflicts in Distributed Heterogeneous Databases: A Case Study," paper presented at the DoD Database Colloquium '93, San Diego, CA, August, 1993.

Kamel, N., "A Data Manipulation Language for ASN.1," working paper, University of Florida, Gainsville, FL, 1993.

Kamel, N., "Markup Languages are a Good Basis for Building Integrated Database/Software Tool Information Resorces," working paper, University of Florida, Gainsville, FL, 1994.

Kim, W., and Seo, J., "Classifying Schematic and Data Heterogeneity in Multidatabase Systems," *IEEE Computer*, v. 24, No. 12, pp. 12-18, 1991.

Larson, J., Navathe, S., and Elmasri, R., "A Theory of Attribute Equivalence in Databases with Application to Schema Integration," *IEEE Transactions on Software Engineering*, v. 15, No. 4, pp. 449-463, 1989.

Liang, J., *A Graphical Environment for Manipulating Molecular Biological Databases Based on ASN.1 Specification,* Master's Thesis, University of Florida, Gainsville, FL, 1993.

National Center for Biotechnology Information, *NCBI Software Development ToolKit User's Manual--Draft Copy*, Version 1.8, August 1, 1993.

Rafii, A., et al. "Integration Strategies in Pegasus Object Oriented Multidatabase System," paper presented at the Proc. of the 25th Hawaii International Conference on System Sciences, v. II, pp. 323-334, 1992.

Sheth, A., and Gala, S., "Attribute Relationships: An Impediment in Automating Schema Integration," paper presented at the Proc. Workshop on Heterogeneous Database Systems, 1989.

Sheth, A., and Larson, J., "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases," *ACM Computing Survey*, Vol. 22, No. 3, Sept., 1990, pp. 183-236.

Thieme, C., and Siebes, A., "An Approach to Schema Integration Based on Transformations and Behaviour," CWI, Amsterdam, The Netherlands.

Young, D., *The X Window System Programming and Applications with Xt*, Prentice Hall, NJ, 1994.

Yu, C., Sun, W., Dao, S., and Keirsey, D., "Determining Relationships Among Attributes for Interoperability of Multi-database Systems," paper presented at the Proc. Int. Workshop on Interoperability in Multidatabase Systems, pp. 251-257, 1991.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center — 2
   Cameron Station
   Alexandria, Virginia 22304-6145

2. Library, Code 52 — 2
   Naval Postgraduate School
   Monterey, California 93943-5101

3. Magdi Kamel, Code SM/Ka — 2
   Department of Systems Management
   Naval Postgraduate School
   Monterey, California 93943-5002

4. James Emery, Code SM/Ey — 1
   Department of Systems Management
   Naval Postgraduate School
   Monterey, California 93943-5002

5. LT Rick Arai, USN, Code 36 — 1
   Naval Postgraduate School
   Monterey, California 93943-5002

6. LT Gino Celia, Jr., USN — 4
   1044 Halsey Dr.
   Monterey, California 93940

7. James Ostell — 2
   National Center for Biotechnology Information
   National Library of Medicine
   National Institutes of Health, Bldg. 38A
   8600 Rockville Pike
   Bethesda, Maryland 20984

8. Mary S. Russo 1
2103 Green Oaks Circle
Rund Rock, Texas 78746

9. Nabil N. Kamel, Code CSE-301 1
Computer and Information Systems Department
University of Florida
Gainsville, Florida 32611